400081

National Semiconductor

Programmable Logic Devices Databook and Design Guide

GAL
ECL PAL
High Density MAPL

PROGRAMMABLE LOGIC DEVICES

DATABOOK AND DESIGN GUIDE

GAL

ECL PAL

High Density MAPL

1993 Edition

PLD Design and Fabrication 1

Low Density GAL and PAL Devices
—Data Sheets and Application Notes 2

High Density MAPL Family
—Data Sheets and Application Notes 3

PLD Development Tools 4

Appendices/Physical Dimensions 5

TRADEMARKS

Following is the most current list of National Semiconductor Corporation's trademarks and registered trademarks.

ABiC™
Abuseable™
Anadig™
APPS™
ASPECT™
Auto-Chem Deflasher™
BCP™
BI-FET™
BI-FET II™
BI-LINE™
BIPLAN™
BLC™
BLX™
BMAC™
Brite-Lite™
BSI™
BSI-2™
CDD™
CIM™
CIMBUS™
CLASIC™
COMBO®
COMBO I™
COMBO II™
COPS™ microcontrollers
CRD™
DA4™
DENSPAK™
DIB™
DISCERN™
DISTILL™
DNR®
DPVM™
E^{2}CMOS™
ELSTAR™
Embedded System
  Processor™
EP™
E-Z-LINK™

FACT™
FACT Quiet Series™
FAIRCAD™
Fairtech™
FAST®
FASTr™
Flash™
GENIX™
GNX™
GTO™
HEX 3000™
HPC™
HyBal™
I^{3}L®
ICM™
Integral ISE™
Intelisplay™
Inter-LERIC™
Inter-RIC™
ISE™
ISE/06™
ISE/08™
ISE/16™
ISE32™
ISOPLANAR™
ISOPLANAR-Z™
LERIC™
LMCMOS™
M^{2}CMOS™
Macrobus™
Macrocomponent™
MAPL™
MAXI-ROM®
Microbus™ data bus
MICRO-DAC™
µtalker™
Microtalker™
MICROWIRE™

MICROWIRE/PLUS™
MOLE™
MPA™
MST™
Naked-8™
National®
National Semiconductor®
National Semiconductor
  Corp.®
NAX 800™
Nitride Plus™
Nitride Plus Oxide™
NML™
NOBUS™
NSC800™
NSCISE™
NSX-16™
NS-XC-16™
NTERCOM™
NURAM™
OPAL™
OXISS™
P^{2}CMOS™
Perfect Watch™
PLAN™
PLANAR™
PLAYER™
Plus-2™
Polycraft™
Power + Control™
POWERplanar™
QS™
QUAD3000™
QUIKLOOK™
RAT™
RIC™
RTX16™
SCX™

SERIES/800™
Series 32000®
Simple Switcher™
SofChek™
SONIC™
SPIRE™
Staggered Refresh™
STAR™
Starlink™
STARPLEX™
ST-NIC™
SuperAT™
Super-Block™
SuperChip™
SuperScript™
SYS32™
TapePak®
TDS™
TeleGate™
The National Anthem®
TLC™
Trapezoidal™
TRI-CODE™
TRI-POLY™
TRI-SAFE™
TRI-STATE®
TROPIC™
Tropic Pele'™
Tropic Reef™
TURBOTRANSCEIVER™
VIP™
VR32™
WATCHDOG™
XMOS™
XPU™
Z STAR™
883B/RETS™
883S/RETS™

ABEL™ is a trademark of Data I/O Corporation.
CUPL® is a registered trademark of Logical Devices.
Data I/O® is a registered trademark of Data I/O Corporation.
Epson® is a registered trademark of Epson America, Inc.
GAL® and UltraMOS® are registered trademarks of Lattice Semiconductor.
Hercules® is a registered trademark of Hercules Computer Technology.
HP® is a registered trademark of Hewlett-Packard Company.
IBM®, PC-AT® and PC-XT® are registered trademarks of International Business Machines Corp.
IFL™ is a trademark of Signetics Semiconductor.
Laserjet™ is a trademark of Hewlett-Packard Company.
Latch-Lock™ is a trademark of Lattice Semiconductor.
MS-DOS® and Windows® are registered trademarks of Microsoft Corporation.
PAL® is a registered trademark of and used under license from Advanced Micro Devices, Inc.
PLDesigner™ is a trademark of Minc Incorporated.
Sun® is a registered trademark of Sun Microsystems.
UNIX® is a registered trademark of AT & T Bell Laboratories.
VAX™ is a trademark of Digital Equipment Corporation.
Viewlogic® is a registered trademark of Viewlogic Systems, Inc.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation 2900 Semiconductor Drive, P.O. Box 58090, Santa Clara, California 95052-8090 1-800-272-9959
TWX (910) 339-9240

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied, and National reserves the right, at any time without notice, to change said circuitry or specifications.

Introduction

This Databook and Design Guide is your complete reference to all product and support information relating to National Semiconductor's programmable logic devices (PLDs) and development tools.

In our new 1992 issue, you will notice that we have expanded our EECMOS product line with the addition of many new GAL® devices, including the new GAL Quiet Series™ family. We have also introduced a brand new family of higher-density EECMOS MAPL™ (Multiple Array Programmable Logic) devices which specifically address the needs of complex state machine and bus interface applications.

To support all of our GAL, ECLPAL and MAPL devices, we have released OPAL™—a low cost, menu driven, open-architecture development package that is available for MS-DOS®, Windows® 3.0 and Sun® workstation platforms.

If you are in need of further information or assistance, please contact your local National Semiconductor sales representative. A detailed list of National Semiconductor sales offices and representatives is given at the back of this book.

National
Semiconductor

Product Status Definitions

Definition of Terms

Data Sheet Identification	Product Status	Definition
Advance Information	Formative or In Design	This data sheet contains the design specifications for product development. Specifications may change in any manner without notice.
Preliminary	First Production	This data sheet contains preliminary data, and supplementary data will be published at a later date. National Semiconductor Corporation reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
No Identification Noted	Full Production	This data sheet contains final specifications. National Semiconductor Corporation reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
Obsolete	Not In Production	This data sheet contains specifications on a product that has been discontinued by National Semiconductor Corporation. The data sheet is printed for reference information only.

National Semiconductor Corporation reserves the right to make changes without further notice to any products herein to improve reliability, function or design. National does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

Selection Tables ix
Ordering Information xi

Section 1 PLD Design and Fabrication

Designing with Programmable Logic 1-3
Programmable Logic Design Methodology 1-11
Fabrication of Programmable Logic 1-21

Section 2 Low Density GAL and PAL Devices

GAL AND PAL DATASHEETS

GAL16V8/A 20-Pin Generic Array Logic Family 2-3
GAL20V8/A 24-Pin Generic Array Logic Family 2-22
GAL16V8QS 20-Pin Generic Array Logic Family 2-40
GAL20V8QS 24-Pin Generic Array Logic Family 2-59
GAL22V10 Generic Array Logic 2-78
GAL20RA10 Generic Array Logic 2-91
GAL6001 Generic Array Logic 2-102
ECL PAL10/10016P8 Combinatorial PAL 2-113
ECL PAL10/10016P8-3 3 ns Combinatorial PAL 2-122
ECL PAL10/10016PE8-3 3 ns Combinatorial PAL 2-127
ECL PAL10/10016P4A Combinatorial PAL 2-132
ECL PAL10/10016P4-2 2 ns Combinatorial PAL 2-136
ECL PAL10/10016C4-2 2 ns Combinatorial PAL 2-141

GAL AND PAL APPLICATION EXAMPLES

1.0 Basic Gates 2-146
2.0 Basic Flip-Flops 2-151
3.0 Quad 4-to-1 Multiplexer 2-156
4.0 Dual 8-to-1 Multiplexer 2-160
5.0 7-Bit Counter with Parallel-Load 2-165
6.0 10-Bit Up/Down Counter 2-169
7.0 8-Bit Barrel Shifter 2-177
8.0 Hexadecimal 7-Segment Display Encoder 2-181
9.0 Dual-Port RAM Controller 2-187
10.0 CPU Board Random Control Logic 2-196

GAL AND PAL APPLICATION NOTES

AN-707 High Speed System Design Using Programmable GAL Devices 2-198
AN-799 PAL to GAL Conversion 2-209
AN-667 A GAL6001-30L Zero Wait State Page Mode Memory System Interface Between the DP8422A and the 68020 2-220
AN-669 A GAL Interface Between Static Random Access Memory (SRAM) and the NSC Raster Graphics Processor (RGP, DP8500) 2-230
AN-618 A GAL Interface for a 25 MHz and above No-Wait State DP8422A/80286 Burst Mode DRAM Memory System 2-237
AN-535 A GAL Interface for a Dual Access DP8422A/68030/74F632 Error Detecting and Correcting Memory System 2-246

Section 3 High Density MAPL Family

MAPL DATASHEETS

MAPL128/MAPL144 Multiple Array Programmable Logic 3-3
MAPL244 Multiple Array Programmable Logic (Preliminary) 3-15
MAPL268 Multiple Array Programmable Logic (Advanced Information) 3-31

Table of Contents (Continued)

Section 3 High Density MAPL Family (Continued)

MAPL APPLICATION EXAMPLES

1.0 MAPL Register Control 3-33
2.0 4-Bit Counter 3-38
3.0 3-Bit Up/Down Counter with 7-Segment Display Output 3-41
4.0 15-Bit Up/Down Counter 3-43
5.0 12-Bit L/R Shift Register 3-48
6.0 High Speed Frame Buffer Control 3-52
7.0 MAPL Divide Down Counter 3-60
8.0 Bus Transaction and DMA Controller 3-67
9.0 MAPL PC Interface Module for the FDDI MAC Layer Evaluation Board 3-78
10.0 PLUS405 to MAPL128 Conversion 3-85

MAPL APPLICATION NOTES

AN-810 Advanced PLD Architectures 3-87
AN-800 A Smart UART Design Implemented in a MAPL128 3-97
AN-801 State Machine Design 3-105
AN-802 MAPL Demonstration Board 3-116

Section 4 PLD Development Tools

OPAL PLD Development Software 4-3
3rd Party Software 4-15
PLD Programming and Testing 4-17

Section 5 Appendices/Physical Dimensions

Appendix A—Boolean Logic Review 5-3
A.1 Basic Operators and Theorems 5-3
A.2 Derivation of a Boolean Expression 5-4
A.3 Minimization 5-6
A.4 K-Map Method 5-7
A.5 Sequential Circuit Elements 5-11
A.6 State Machine Fundamentals 5-12
A.7 Avoiding Logic Hazards 5-14
Appendix B—Theory of PLD Testing 5-16
B.1 Testing Methods 5-16
B.2 Test Generation Techniques 5-17
B.3 Response Evaluation Techniques 5-27
Physical Dimensions 5-32
Bookshelf
Distributors

Alpha-Numeric Index

AN-535 A GAL Interface for a Dual Access DP8422A/68030/74F632 Error Detecting and Correcting Memory System 2-246

AN-618 A GAL Interface for a 25 MHz and above No-Wait State DP8422A/80286 Burst Mode DRAM Memory System 2-237

AN-667 A GAL6001-30L Zero Wait State Page Mode Memory System Interface Between the DP8422A and the 68020 2-220

AN-669 A GAL Interface Between Static Random Access Memory (SRAM) and the NSC Raster Graphics Processor (RGP, DP8500) 2-230

AN-707 High Speed System Design Using Programmable GAL Devices 2-198

AN-799 PAL to GAL Conversion 2-209

AN-800 A Smart UART Design Implemented in a MAPL128 3-97

AN-801 State Machine Design 3-105

AN-802 MAPL Demonstration Board 3-116

AN-810 Advanced PLD Architectures 3-87

GAL16V8-7L 20-Pin Generic Array Logic Family 2-3

GAL16V8-10L 20-Pin Generic Array Logic Family 2-3

GAL16V8-20L 20-Pin Generic Array Logic Family 2-3

GAL16V8-25L 20-Pin Generic Array Logic Family 2-3

GAL16V8-25Q 20-Pin Generic Array Logic Family 2-3

GAL16V8A-12L 20-Pin Generic Array Logic Family 2-3

GAL16V8A-15L 20-Pin Generic Array Logic Family 2-3

GAL16V8QS-15L 20-Pin Generic Array Logic Family 2-40

GAL16V8QS-15Q 20-Pin Generic Array Logic Family 2-40

GAL16V8QS-20L 20-Pin Generic Array Logic Family 2-40

GAL16V8QS-25L 20-Pin Generic Array Logic Family 2-40

GAL16V8QS-25Q 20-Pin Generic Array Logic Family 2-40

GAL20RA10-15 Generic Array Logic 2-91

GAL20RA10-20 Generic Array Logic 2-91

GAL20RA10-25 Generic Array Logic 2-91

GAL20V8-7L 24-Pin Generic Array Logic Family 2-22

GAL20V8-10L 24-Pin Generic Array Logic Family 2-22

GAL20V8-20L 24-Pin Generic Array Logic Family 2-22

GAL20V8-25L 24-Pin Generic Array Logic Family 2-22

GAL20V8-25Q 24-Pin Generic Array Logic Family 2-22

GAL20V8A-12L 24-Pin Generic Array Logic Family 2-22

GAL20V8A-15L 24-Pin Generic Array Logic Family 2-22

GAL20V8QS-15L 24-Pin Generic Array Logic Family 2-59

GAL20V8QS-15Q 24-Pin Generic Array Logic Family 2-59

GAL20V8QS-20L 24-Pin Generic Array Logic Family 2-59

GAL20V8QS-25L 24-Pin Generic Array Logic Family 2-59

GAL20V8QS-25Q 24-Pin Generic Array Logic Family 2-59

GAL22V10-15 Generic Array Logic 2-78

GAL22V10-20 Generic Array Logic 2-78

GAL22V10-25 Generic Array Logic 2-78

GAL22V10-30 Generic Array Logic 2-78

GAL6001 Generic Array Logic 2-102

MAPL128-33 Multiple Array Programmable Logic 3-3

MAPL128-40 Multiple Array Programmable Logic 3-3

MAPL144-33 Multiple Array Programmable Logic (Preliminary) 3-3

MAPL144-40 Multiple Array Programmable Logic (Preliminary) 3-3

MAPL144-45 Multiple Array Programmable Logic (Preliminary) 3-3

MAPL244-33 Multiple Array Programmable Logic (Preliminary) 3-15

Alpha-Numeric Index (Continued)

MAPL244-40 Multiple Array Programmable Logic (Preliminary) 3-15
MAPL244-50 Multiple Array Programmable Logic (Preliminary) 3-15
MAPL268 Multiple Array Programmable Logic (Advanced Information) 3-31
OPAL PLD Development Software 4-3
PAL1016C4-2 ECL 2 ns Combinatorial PAL 2-141
PAL1016P4-2 ECL 2 ns Combinatorial PAL 2-136
PAL1016P4A ECL 4 ns Combinatorial PAL 2-132
PAL1016P8 ECL Combinatorial PAL 2-113
PAL1016P8-3 ECL 3 ns Combinatorial PAL 2-122
PAL1016PE8-3 ECL 3 ns Combinatorial PAL 2-127
PAL10016C4-2 ECL 2 ns Combinatorial PAL 2-141
PAL10016P4-2 ECL 2 ns Combinatorial PAL 2-136
PAL10016P4A ECL 4 ns Combinatorial PAL 2-132
PAL10016P8 ECL Combinatorial PAL 2-113
PAL10016P8-3 ECL 3 ns Combinatorial PAL 2-122
PAL10016PE8-3 ECL 3 ns Combinatorial PAL 2-127

National Semiconductor

Product Line Overview

1.0 Selection Tables

The product selection tables which follow are organized by technology group (EECMOS and ECL), then by "family", and finally by "series" within each family. The term "family" refers to a set of one or more "device types" which are variations on the same basic architecture. The term "device type" refers to a specific device architecture (regardless of performance characteristics). The term "series" refers to a particular speed/power version in which the devices of a PLD family are available. Thus as technology advancements provide for improved speed/power performance, a new series is added to selected product families.

TABLE I. Programmable Logic Product Selection Guide

Family and Series	Part Numbers	t_{PD} (max) (Note 1)	I_{CC} (max)	Outputs		Page
				Combinatorial	Registered	
EECMOS						
20-Pin Quiet Series Generic Array Logic	GAL16V8QS-15L	15	90	—	8	2-40
	GAL16V8QS-15Q	15	55	—	8	2-40
	GAL16V8QS-20L	20	90	—	8	2-40
	GAL16V8QS-25L	25	90	—	8	2-40
	GAL16V8QS-25Q	25	55	—	8	2-40
24-Pin Quiet Series Generic Array Logic	GAL20V8QS-15L	15	90	—	8	2-59
	GAL20V8QS-15Q	15	55	—	8	2-59
	GAL20V8QS-20L	20	90	—	8	2-59
	GAL20V8QS-25L	25	90	—	8	2-59
	GAL20V8QS-25Q	25	55	—	8	2-59
20-Pin Generic Array Logic	GAL16V8-7	7.5	115	—	8	2-3
	GAL16V8-10	10	115	—	8	2-3
	GAL16V8A-12	12	90	—	8	2-3
	GAL16V8A-15	15	90	—	8	2-3
	GAL16V8-20L	20	90	—	8	2-3
	GAL16V8-25L	25	90	—	8	2-3
	GAL16V8-25Q	25	55	—	8	2-3
24-Pin Generic Array Logic	GAL20V8-7	7.5	115	—	8	2-22
	GAL20V8-10	10	115	—	8	2-22
	GAL20V8A-12	12	90	—	8	2-22
	GAL20V8A-15	15	90	—	8	2-22
	GAL20V8-20L	20	90	—	8	2-22
	GAL20V8-25L	25	90	—	8	2-22
	GAL20V8-25Q	25	55	—	8	2-22
	GAL22V10-15	15	130	—	10	2-78
	GAL22V10-20	20	150	—	10	2-78
	GAL22V10-25	25	130	—	10	2-78
	GAL22V10-30	30	150	—	10	2-78
	GAL20RA10-15	15	100	—	10	2-91
	GAL20RA10-20	20	120	—	10	2-91
	GAL20RA10-25	25	100	—	10	2-91
	GAL6001-30L	30	150	—	10	2-102

Note 1: Maximum t_{PD} for combinatorial outputs (commercial operating range). Denotes characteristic speed of family where product has all non-registered outputs.

TABLE I. Programmable Logic Product Selection Guide (Continued)

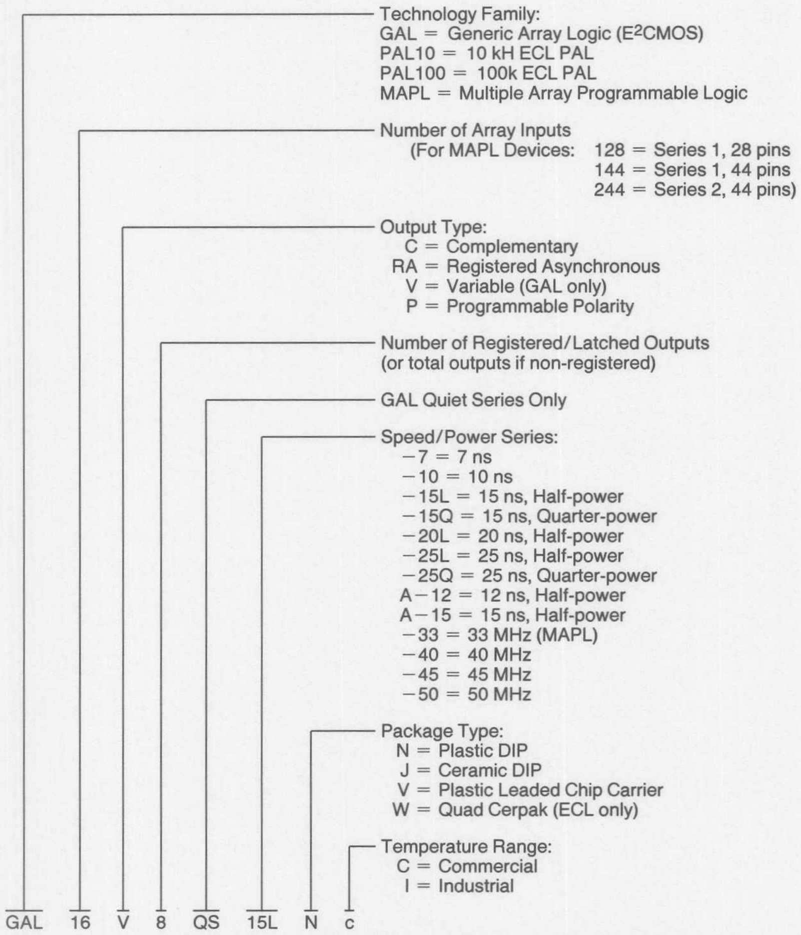
Family and Series	Part Numbers	t_{PD} (max) (Note 1)	I_{CC} (max)	Outputs		Page
				Combinatorial	Registered	
ECL PAL						
Combinatorial	PAL1016P8	6	−240	8	—	2-113
	PAL10016P8	6	−240	8	—	2-113
	PAL1016P8-3	3	−230	8	—	2-122
	PAL10016P8-3	3	−230	8	—	2-122
	PAL1016PE8-3	3	−230	8	—	2-127
	PAL10016PE8-3	3	−230	8	—	2-127
	PAL1016P4A	4	−220	4	—	2-132
	PAL10016P4A	4	−220	4	—	2-132
	PAL1016P4-2	2.5	−220	4	—	2-136
	PAL10016P4-2	2.5	−220	4	—	2-136
	PAL1016C4-2	2	−220	4	—	2-141
	PAL10016C4-2	2	−220	4	—	2-141

Note 1: Maximum t_{PD} for combinatorial outputs (commercial operating range). Denotes characteristic speed of family where product has all non-registered outputs.

Family and Series	Part Numbers	$F_{(max)}$	I_{CC} (max)	Outputs		Page
				Combinatorial	Registered	
HIGH DENSITY EECMOS						
Multiple Array Programmable Logic	MAPL128-40	40	110	—	16	3-3
	MAPL128-33	33	110	—	16	3-3
	MAPL144-45	45	125	—	24	3-3
	MAPL144-40	40	125	—	24	3-3
	MAPL144-33	33	125	—	24	3-3
	MAPL244-50	50	180	8	24	3-15
	MAPL244-40	40	180	8	24	3-15
	MAPL244-33	33	180	8	24	3-15

2.0 Ordering Information

The ordering information diagram below defines the product-number nomenclature used throughout National's programmable logic product line. This nomenclature is based on that used by the original industry-standard PAL products, and are therefore very similar to the product numbers used by other PLD manufacturers. Refer to the corresponding "Ordering Information" diagrams in the individual product datasheets to determine the valid combinations of attributes describing actual PLD products.



Technology Family:
GAL = Generic Array Logic (E^2CMOS)
PAL10 = 10 kH ECL PAL
PAL100 = 100k ECL PAL
MAPL = Multiple Array Programmable Logic

Number of Array Inputs
(For MAPL Devices: 128 = Series 1, 28 pins
144 = Series 1, 44 pins
244 = Series 2, 44 pins)

Output Type:
C = Complementary
RA = Registered Asynchronous
V = Variable (GAL only)
P = Programmable Polarity

Number of Registered/Latched Outputs
(or total outputs if non-registered)

GAL Quiet Series Only

Speed/Power Series:
−7 = 7 ns
−10 = 10 ns
−15L = 15 ns, Half-power
−15Q = 15 ns, Quarter-power
−20L = 20 ns, Half-power
−25L = 25 ns, Half-power
−25Q = 25 ns, Quarter-power
A−12 = 12 ns, Half-power
A−15 = 15 ns, Half-power
−33 = 33 MHz (MAPL)
−40 = 40 MHz
−45 = 45 MHz
−50 = 50 MHz

Package Type:
N = Plastic DIP
J = Ceramic DIP
V = Plastic Leaded Chip Carrier
W = Quad Cerpak (ECL only)

Temperature Range:
C = Commercial
I = Industrial

FIGURE 1-1. PLD Part-Number Nomenclature and Ordering Information

Section 1

PLD Design and Fabrication

Section 1 Contents

Designing with Programmable Logic 1-3
Programmable Logic Design Methodology 1-11
Fabrication of Programmable Logic 1-21

National Semiconductor

Designing with Programmable Logic

Programmable logic has evolved over the last decade into a design tool permitting digital logic designs with a minimal number of packages and a maximum of flexibility. The key to PLDs is the use of embedded programmable cells which allow logic components to be configured into specific designs in the field. This permits logic consolidation with quick implementation and equally quick design revision often without board layout changes.

While Programmable Logic Devices (PLDs) do not offer the density of standard VLSI or custom circuits, they are far more flexible than the former and more cost-efficient than the latter. They have found extensive use in varied applications. They are both inexpensive and space-saving in replacing less efficient "glue logic", which was one of the more popular original uses of PLDs. But they are also capable of efficiently implementing complex functions and state machines.

1.0 Background to PLDs

The use of programmable logic in digital design began with the diode matrix with aluminum fuses at the crosspoints in the early 1960's. This evolved into the PROM through the addition of a decoder at the inputs. The result was an addressable memory which could also be seen as a universal logic device with a fixed AND matrix (the decoder) feeding a programmable OR matrix (the diode array). A representative PROM circuit is shown in *Figure 1(b)* as implementing, for example, a simple set of equations given in *Figure 1(a)*. The disadvantage of a PROM used as a logic device derives from its universality. The number of product terms available is 2^n, where n is the number of input variables. Each additional variable (input pin) doubles the size of the matrix. As a result, commercial PROMs offer limited input (i.e. address pins). This approach rapidly degrades performance due to the decode logic and the array dimensions required, and increases cost through inefficient use of silicon. Many logic applications require more inputs, but not the flexibility of a full decoder.

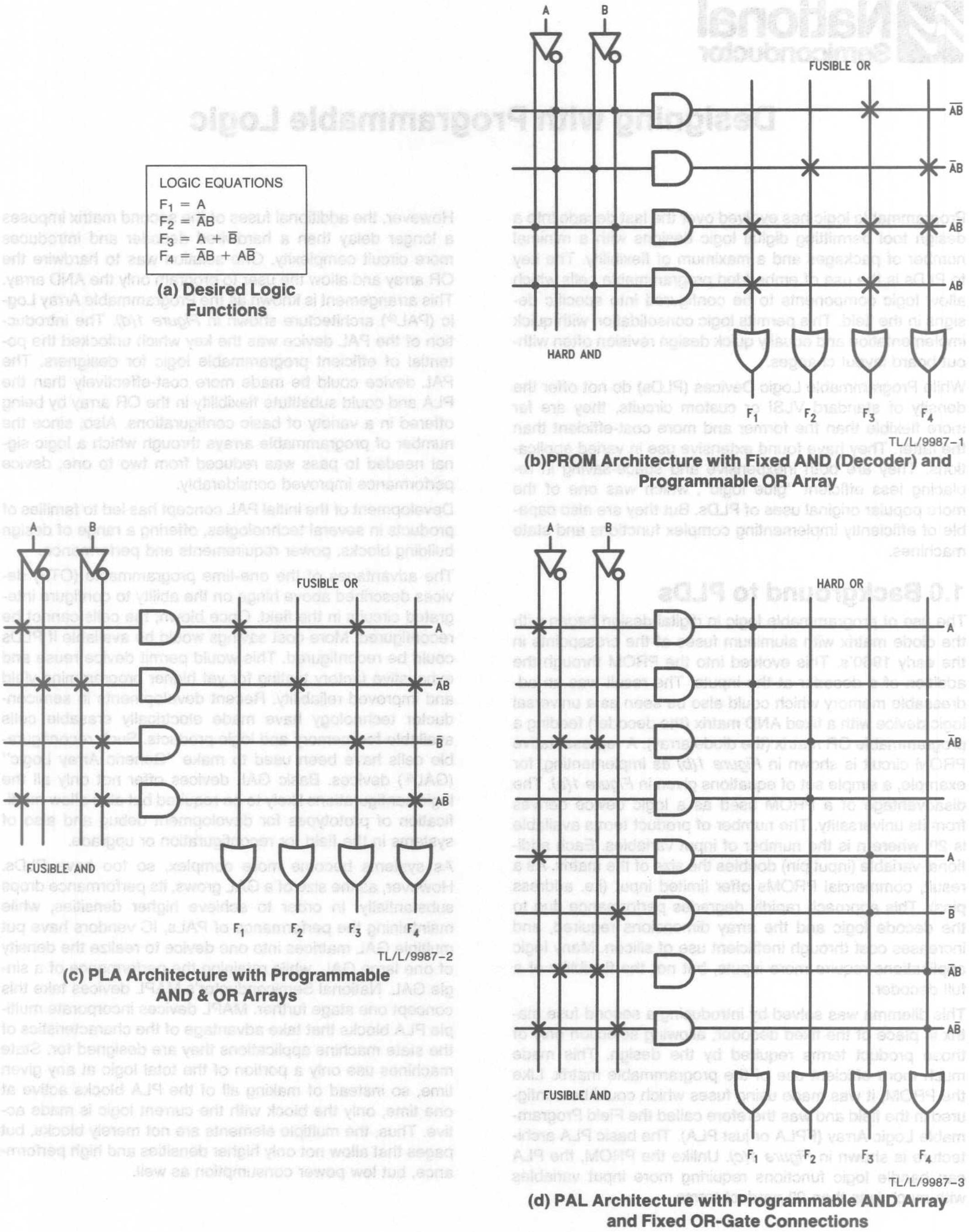
This dilemma was solved by introducing a second fuse matrix in place of the fixed decoder, allowing selection only of those product terms required by the design. This made much more efficient use of the programmable matrix. Like the PROM, it was made using fuses which could be configured in the field and was therefore called the Field Programmable Logic Array (FPLA or just PLA). The basic PLA architecture is shown in *Figure 1(c)*. Unlike the PROM, the PLA can handle logic functions requiring more input variables with much less than 2^n product terms.

However, the additional fuses of the second matrix imposes a longer delay than a hardwired decoder and introduces more circuit complexity. One solution was to hardwire the OR array and allow the user to program only the AND array. This arrangement is known as the Programmable Array Logic (PAL®) architecture shown in *Figure 1(d)*. The introduction of the PAL device was the key which unlocked the potential of efficient programmable logic for designers. The PAL device could be made more cost-effectively than the PLA and could substitute flexibility in the OR array by being offered in a variety of basic configurations. Also, since the number of programmable arrays through which a logic signal needed to pass was reduced from two to one, device performance improved considerably.

Development of the initial PAL concept has led to families of products in several technologies, offering a range of design building blocks, power requirements and performance.

The advantages of the one-time programmable (OTP) devices described above hinge on the ability to configure integrated circuits in the field. Once blown, the cells cannot be reconfigured. More cost savings would be available if PLDs could be reconfigured. This would permit device reuse and exhaustive factory testing for yet higher programming yield and improved reliability. Recent developments in semiconductor technology have made electrically erasable cells available for memory and logic products. Such reconfigurable cells have been used to make "Generic Array Logic" (GAL®) devices. Basic GAL devices offer not only all the logic configurations likely to be required but also allow modification of prototypes for development debug and also of systems in the field for reconfiguration or upgrade.

As systems become more complex, so too have PLDs. However, as the size of a GAL grows, its performance drops substantially. In order to achieve higher densities, while maintaining the performance of PALs, IC vendors have put multiple GAL matrices into one device to realize the density of one large GAL, while retaining the performance of a single GAL. National Semiconductor's MAPL devices take this concept one stage further. MAPL devices incorporate multiple PLA blocks that take advantage of the characteristics of the state machine applications they are designed for. State machines use only a portion of the total logic at any given time, so instead of making all of the PLA blocks active at one time, only the block with the current logic is made active. Thus, the multiple elements are not merely blocks, but pages that allow not only higher densities and high performance, but low power consumption as well.



LOGIC EQUATIONS

$F_1 = A$

$F_2 = \bar{A}B$

$F_3 = A + \bar{B}$

$F_4 = \bar{A}B + A\bar{B}$

(a) Desired Logic Functions

TL/L/9987–1

(b) PROM Architecture with Fixed AND (Decoder) and Programmable OR Array

TL/L/9987–2

(c) PLA Architecture with Programmable AND & OR Arrays

TL/L/9987–3

(d) PAL Architecture with Programmable AND Array and Fixed OR-Gate Connections

FIGURE 1. Comparison of Programmable Logic Basic Architectures

2.0 Design Advantages

Digital logic designers have always worked under constraints. Reduction of system size and cost demand efficient, compact designs. System reliability forces designers to compromise between evolving solutions and existing proven methods. Future revisions demand a degree of flexibility which must be anticipated. Yet the systems themselves increase in complexity, components sophistication requires ever more sophisticated tools and conceiving the optimal design for so many parameters requires a range of skills which must constantly be developed.

PLDs do not solve all of these problems. But they do provide a method of dealing with some of the major issues in an effective way by providing a uniform methodology. This section discusses some of the major advantages available to designers through the use of PLDs.

SIMPLIFIED SYSTEM DESIGN

The semi-custom approach of PLDs allows the user to specify exactly the functions which will be implemented in the logic. This avoids the problem of interconnecting various SSI components to achieve the same result. At the same time, PLDs offer speed advantages through reduction of interconnects. The methodology becomes one of writing the equations for the desired function with the help of the software tools and using such equations to configure the appropriate devices. This methodology accelerates both the conception and implementation of the design. Since the software tools available handle all the details regarding device configuration, the designer is left free to focus on the design of the application itself. An additional benefit is that many of the changes which usually need to be incorporated in a design after implementation can often be accommodated by altering the PLD's internal configuration, thus avoiding rewiring of prototype and printed circuit boards.

INCREASED FUNCTIONAL DENSITY

Despite the development of LSI and VLSI devices which package an amazing amount of logic on a single chip, system designers have still had to contend with the power, space and drive problems associated with a myriad of SSI/MSI packages used either for "glue" or for specific design requirements not available in off-the-shelf parts. Ordinarily, this will decrease the functional density.

PLDs, however, offer a compact solution with high functionality and less waste in I/O and interconnect lines, so that functional density can approach that of custom logic without the associated engineering charges. On the other hand, the combination of several functions on a single chip reduces power as well as space and has the added benefit of boosting system performance through a reduction of interconnects.

3.0 Manufacturing Advantages

While PLDs offer a number of advantages over SSI/MSI for the designer, there are a number of considerations which only become apparent when system volume production is examined. These include:

- Cost of Inventory
- Cost of Ownership
- Cost of Upgrades/Modifications
- Reliability

COST OF INVENTORY

A hidden cost associated with many designs is the inventory of parts required in order to sustain it in production. PLDs are able to reduce this cost by offering a substantial reduction in the number of different part types which are otherwise required to build a given system using standard logic parts. This is particularly true for GAL devices. Users may find that the inventory cost advantages of GAL devices tend to offset the slight difference in price of GAL over standard PAL devices.

COST OF OWNERSHIP

The cost of ownership of a particular part is more subtle than the simple price at which it is available on the market. In fact, the cost of ownership includes the cost of those devices which fail and which must be replaced. This cost increases dramatically as the discovery of failures occurs later in the production process.

The additional cost of a failed part at incoming inspection is relatively minor. However, PLDs must be programmed to the user's pattern before any meaningful functional testing can be done by the user. Therefore, the first detectable device failure for a PLD will be a programming failure detected during device verification on the programming equipment. This is the most frequent of all failure modes for PLDs. Electrically-erasable GAL and MAPL devices have a much higher factory testability and an inherently more reliable programming technology. This gives these advanced technology products a strong advantage at this early stage of production.

Once devices have been verified, functional testing can be performed while still loaded in the programming unit or on production IC testers. Otherwise, the device functionality is tested in-circuit. Both involve further production costs which contribute to cost of ownership, particularly if the device is already soldered in place.

Failures beyond this may occur at the board or system level. This sometimes occurs despite testing at previous stages and happens to standard non-programmable products as much as to PLDs. Most are detected in production, but are increasingly costly to correct.

The final location at which device failure over product lifetime can occur is in the field. Field failures are attributed primarily to device reliability. Since field failures have the highest associated cost, National Semiconductor performs extensive reliability testing on all PLD products, processes and packages.

COST OF UPGRADES AND MODIFICATIONS

Unlike standard SSI/MSI, PLDs offer some degree of flexibility in permitting alterations to a circuit which is already in production. At its simplest level, all PLDs permit some degree of reconfiguration within the existing printed circuit board and device pinout. Even if the original one-time-programmable (OTP) PLD cannot be reconfigured, subsequent production can alter the circuit by altering the fuse map without any other changes. Where the change is more dramatic, the use of a pin-compatible GAL device may still offer a change which requires no circuit board alterations. Field alterations are then limited to replacing a device and do not require the standard time-consuming cut-and-jumper approach.

HIGHER RELIABILITY

The higher levels of integration associated with contemporary digital design have brought reliability and testability to the fore as issues in system design. Increased system and integrated circuit complexity have made it crucial that an acceptable design debug and system test methodology be included as part of the development process. PLDs help in this by being both flexible and versatile in design debug, particularly reconfigurable EECMOS devices.

It has also been shown that system reliability is a function of the number of components used and the pins and wiring necessary to interconnect them. The decreased package count over standard SSI/MSI devices through the use of PLDs therefore helps maximize system reliability.

Recent advances in PLD circuit design and processing technology have greatly improved the long-term reliability of both OTP and reprogrammable EECMOS products. Sophisticated test circuitry built into these products allows increased testability of on-chip circuit elements and programming cells to ensure long-term reliability.

4.0 Alternative Methodologies

Since the introduction of LSI, there has been a growing "complexity gap" between high-density, high-functionality devices and the low-density device available to interconnect them or provide non-standard design alternatives. PLDs are emerging to bridge that gap.

STANDARD LSI

Advances in this area have been made at the cost of device flexibility and support software complexity. A hidden disadvantage has been the need to interface such devices into a given system, often resulting in a disproportionately high package count and power supply problems. PLDs can be used for package reduction of the inevitable "glue logic" around LSI applications. But also, frequently design requirements may be for a controller which doesn't require microprocessor complexity or cannot accept the slower microprocessor speeds. While standard LSI attempts to be a universal solution, the system under design may require a much simpler solution and/or special functions not provided in available LSI. PLDs can often provide a more appropriate, higher-performance, cost-effective and compact design in such cases.

STANDARD SSI/MSI

While having mounting competition from other design methodologies, SSI/MSI logic continues to be used in many designs where specialized functions are required in lower production volumes. PLDs offer a more effective means of implementing these functions in all but the most trivial examples because of their ability to reduce package count, increase performance, offer design support tools and simplify revisions and upgrades. Where systems require absolute minimum delays through short logic paths or standard logic functions, standard SSI/MSI devices are often an indispensable solution. However, in most other "glue logic" applications, the long-term efficiency and flexibility of PLDs tend to outweigh any initial parts cost savings derived from using SSI/MSI, especially when design revision is considered.

FULL CUSTOM

These provide an excellent solution to well-defined, low-to-medium complexity logic which is expected to be produced in very high volume. The risks involved in committing both the time and money mean it is seldom used in practice unless extreme performance/density is required or extreme high volume is expected.

SEMI-CUSTOM ASIC's

The success of this semi-custom approach depends on the efficiency of use of the gates available. This requires skillful partitioning of the logic and careful selection of the gate array. It has far less development time and cost associated with it than full custom, but fixed costs must still be considered, along with the difficulty of correcting any problem in the logic once committed to silicon.

Even though prototype development time for gate arrays has been significantly reduced over the past few years, alomost any redesign requirements can imply mask revision and have a devastating effect on system introduction.

COMPLEX PLSs/FPGAs

These devices incorporate multiple arrays or logic blocks into one device. Both complex PLDs and Field Programmable Gate Arrays (FPGAs) are described here together since the distinction between the two is diminishing and since they address the same market. These devices provide the programmability of PAL and GAL devices with the densities of custom and semi-custom ASICs. While the part cost of these devices is greater than that of ASICs, they are much less expensive and much quicker to design with. Thus, as time to market continues to increase in importance, complex PLDs and FPGAs become more and more attractive. See the datasheet section of this databook for information about National Semiconductor's MAPL complex PLDs.

5.0 PLD Architecture Overview

This section describes some of the basic architectural features found in the various PAL-like devices (including the GAL devices). For more detailed descriptions of the architectures of specific devices, refer to the appropriate datasheet.

LOGIC ARRAY STRUCTURE

The PAL architecture is based on a single programmable AND-gate array with fixed OR-gate connections. The AND array consists of a number of "input lines" running in one dimension across a number of "product-term lines" running in the orthogonal dimension with programmable interconnection cells at all intersections. For every input signal (logical input variable) applied to the array, a true and complement pair of input lines are provided.

A product term is satisfied (logically true) while all input lines connected to it (via programmable cells) are high. If neither the true nor complement of an array input is connected to a product line, then that array input represents a "don't care" value with respect to that product term.

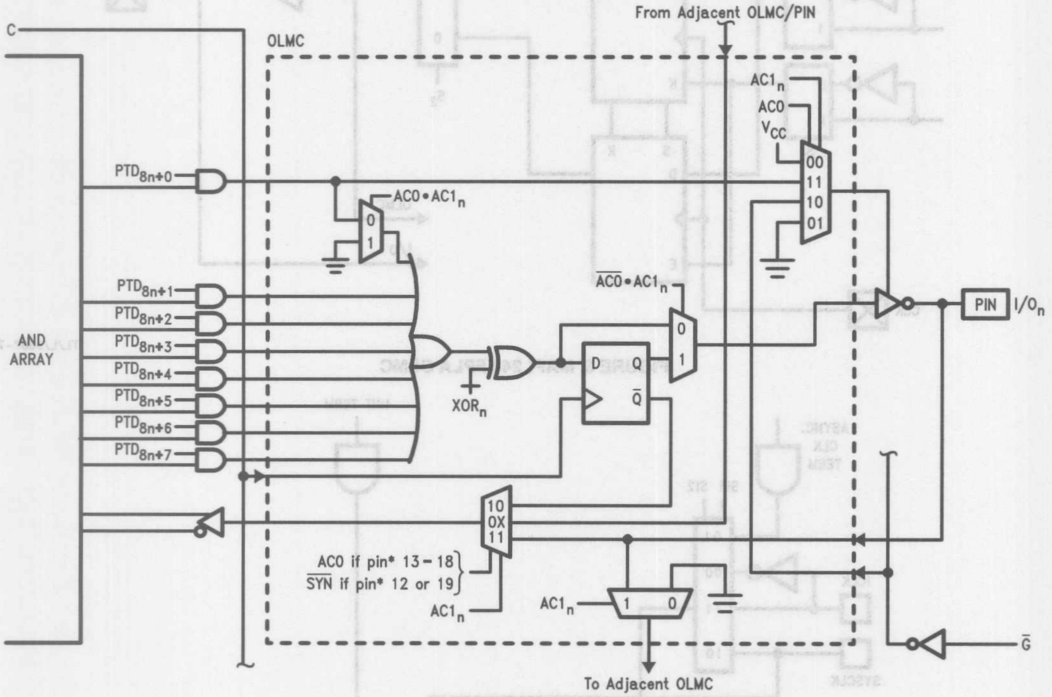
In all PAL-based devices covered in this book, all product terms are dedicated to specific device outputs. The number of product terms allocated to each output logic function varies from device to device.

ADVANCED MACROCELL

GAL devices use additional programming cells to redirect their output logic paths to emulate a wide variety of TTL PAL architectures, plus other original configurations. These "architecture cells" select registered vs. combinatorial outputs, TRI-STATE® control signals, I/O feedback paths, and output polarity. The term "Output Logic Macrocell" (OLMC) is commonly used to describe such sophisticated peripheral logic which is user-configurable by means of programmable architecture switches. *Figure 2* shows the complete logic schematic of a GAL OLMC. Fortunately, the logic paths and architecture switches are automatically configured by design development software according to the designer's familiar logic equations.

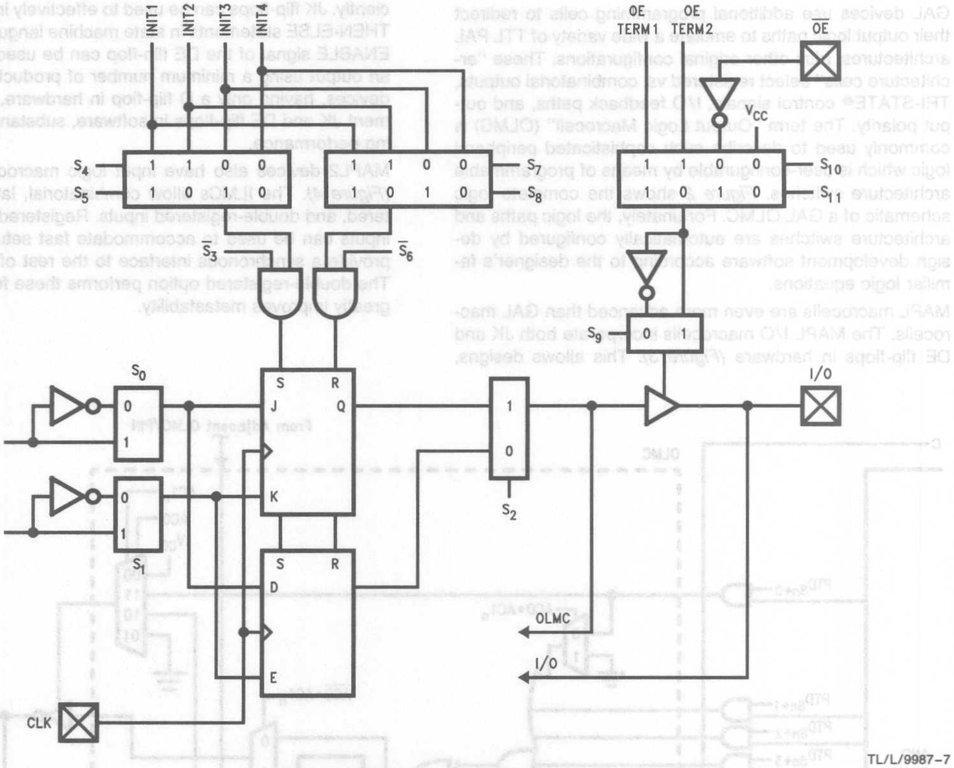
MAPL macrocells are even more advanced than GAL macrocells. The MAPL I/O macrocells incorporate both JK and DE flip-flops in hardware *(Figure 3)*. This allows designs, especially state machines, to be implemented most efficiently. JK flip-flops can be used to effectively implement IF-THEN-ELSE statements in state machine language, and the ENABLE signal of the DE flip-flop can be used to "freeze" an output using a minimum number of product terms. GAL devices, having only a D flip-flop in hardware, must implement JK and DE flip-flops in software, substantially impacting performance.

MAPL2 devices also have input logic macrocells (ILMCs) *(Figure 4)*. The ILMCs allow combinatorial, latched, registered, and double-registered inputs. Registered and latched inputs can be used to accommodate fast setup times and provide a synchronous interface to the rest of the system. The double-registered option performs these functions and greatly improves metastability.



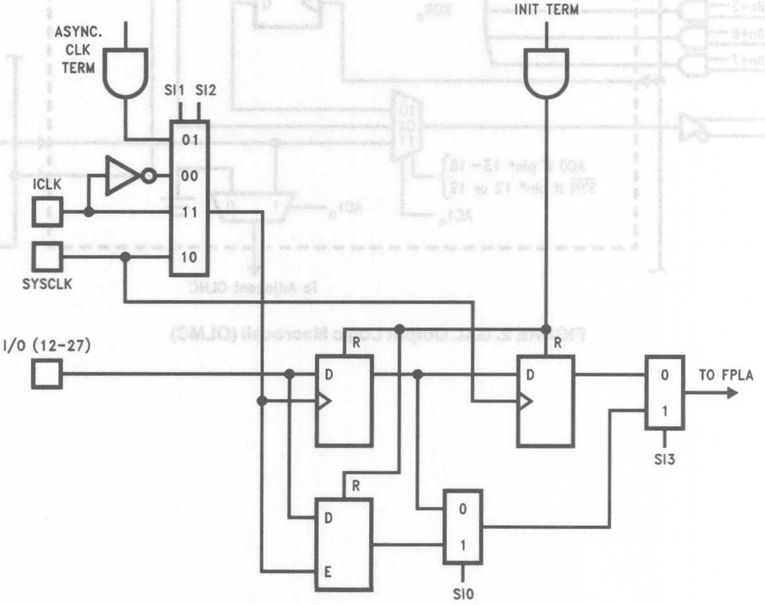
TL/L/9987–6

FIGURE 2. GAL Output Logic Macrocell (OLMC)



TL/L/9987-7

FIGURE 3. MAPL244 FPLA OLMC



TL/L/9987-8

FIGURE 4. MAPL244 ILMC

6.0 Programmable Logic from National

National Semiconductor offers a broad range of products in the PLD field.

ECL PAL PRODUCTS

A large range of the traditional PAL architectures are available in ECL for those system designers taking advantage of this technology. In very high speed applications where ECL is typically used, logic optimization becomes crucial and the delays involved in off-chip wiring become more pronounced as a fraction of the total delay. The SSI/MSI logic families available in ECL are not as robust as those in TTL and tend to consume larger system board areas in implementation. Both problems can be more effectively reduced through the use of ECL PLDs. Consolidating logic into PLDs moves local logic interconnections on-chip eliminating the associated wire delays. Also, by reducing board area consumed by the logic implemented in the PLD, remaining on-board logic becomes more compact thereby reducing the trace lengths of interconnections among surrounding logic. This adds to the speed advantage. Reducing required board area also allows the system designer to increase the amount of on-board logic.

EECMOS GAL PRODUCTS

The CMOS technology offers advantages over standard TTL in some applications. Modern CMOS technologies require considerably less power with little speed penalty. The recent developments in electrically erasable CMOS devices (EECMOS) offer additional advantages in flexibility and testability. Using a cell which can be reprogrammed multiple times, a GAL device offers increased ease in system prototyping.

A GAL device also has a powerful feature in its reconfigurable output macrocells. This permits it to replace a variety of more conventional PAL devices, thus reducing inventory requirements. The intrinsic higher device reliability and ability to be fully tested in the factory with guaranteed yields of 100% can replace customer incoming device inspection.

EECMOS MAPL PRODUCTS

National Semiconductor's new family of high density Multiple Array Programmable Logic (MAPL) devices solve the density limitations of standard PAL, GAL and PLA devices, without the development cost, effort and performance problems associated with FPGAs. The MAPL family is ideally suited to state machine designs and offers considerable performance advantages over other programmable logic solutions. MAPL devices can be configured to implement a superset of many current PLD architectures including:

— Field Programmable Logic Sequences (FPLS)

— Registered PAL/GAL devices

— Low density gate arrays/FPGAs

— Programmable microsequences/microcontrollers

— Registered PROMs

The MAPL family consists of two series, MAPL1 and MAPL2. Both incorporate multiple Field Programmable Logic Arrays (FPLA) which are fully interconnected to resemble one large continuous FPLA. By internally partitioning the arrays in this manner, MAPL is able to achieve up to 45 MHz true system performance with the power consumption equal to that of just a single EECMOS GAL device.

PRE-PROGRAMMED PRODUCTS

As the volume of system production increases and the confidence in the design solidifies, it may make economic sense to consider using one of the pre-programmed device programs from National. For production flows which require quantities of PLDs which cannot be programmed cost-effectively at the customer site, National Semiconductor offers a program which ships GAL devices already programmed to customers for immediate inclusion in the system.

This program avoids the manufacturing costs associated with device programming and related programming failures. It also considerably enhances device reliability due to the thorough factory test performed on functional devices before shipment using the customer's test patterns. Because of the resulting increase in product quality levels, further manufacturing costs may be saved by eliminating incoming component inspection.

7.0 Design Development Tools

Design development with PLDs involves the use of a few tools which assist both in the design conception and implementation stages. Information is widely available describing in detail the usage, features, and issues involved in selecting and using these tools. Competition in the PLD support tools market has considerably reduced the investment necessary to acquire the appropriate hardware and software tools.

PROGRAMMING HARDWARE

PLDs are typically configured on a piece of hardware known as a programmer. Most of today's programmers consist of a stand-alone box housing universal pin-driver electronics and control circuitry, connected to a computer software platform via a serial or parallel link. Device selection and programming control is executed from the computer. Most PLDs can be programmed in a single DIP socket mounted directly on the box. Adapters are often used for SMD devices.

The selection of programmers has widened in recent years, and now includes an extensive variety of both "universal", and device-specific programmers. Many manufacturers worldwide offer programmers, either as a stand-alone system, or as an add-in to an existing PC. Programmers are available in a wide variety of complexity, capabilty, and cost.

Most universal programmers are capable of programming a wide variety of PLD products when equipped with the necessary adapter or software cartridge/diskette. Often the very latest PLD devices can be supported with an update available from the programmer manufacturer. Device-specific programmers are often available for new-technology, or high complexity devices (such as some FPGAs) which require unique, or complex support features not readily available on standard programmers.

The bulk of PLDs today are programmed by downloading a JEDEC file (generated from a PLD development software tool) into the programmer, then programming the device. The JEDEC file represents the fuse map for a specific PLD, and conforms to an industry-wide standard which ensures that files generated by virtually any software tool can be used to program PLDs an virtually any programmer.

National maintains a program of evaluation and certification for programmers supporting NSC devices. All "approved" programmers must meet stringent criteria designed to ensure the highest quality programming results. National strongly recommends using only "approved" programmers for both engineering and production.

For more detailed information see Section 4.

DEVELOPMENT SOFTWARE

Software tools allow the PLD designer to use commonly available computers to enter PLD designs, compile and test the design, select appropriate PLD device types, and create the JEDEC file for downloading to a programmer. The software allows designs to be entered in a variety of ways, including Boolean equations and high-level language, and may include advanced features such as automatic logic reduction and device independence. Virtually all software tools support most common PLDs, and some will also support less common types.

Development software packages are available from a variety of manufacturers worldwide, and offer a range of features, complexity, and cost.

NATIONAL SEMICONDUCTOR OPAL™ SOFTWARE

The OPAL software package is a comprehensive PLD (Programmable Logic Device) development design tool offered by National Semiconductor. It supports state machine, truth table and Boolean equation entry, as well as optimization, verification and implementation in a wide variety of PLDs.

The OPAL software package consists of: a graphical shell environment, executable modules, a graphical simulation package, a device library file, examples and an overall demonstration of the software package.

OPAL is an open architecture language. This means the software is modularized so that there is a standard interface format between modules. These standard interface formats allow the modules to communicate with 3rd party software, thereby letting the user use existing, familiar software tools in conjunction with OPAL. While OPAL is a powerful standalone PLD development tool, it complements the user's existing tools (rather than superceding them) to form an even more powerful and flexible toolbox.

OTHER ASSEMBLERS/COMPILERS

High-level assemblers provide many automatic features that are not found in other assemblers. They are usually device and manufacturer-independent and some tools may be usable with other software packages. While they are still capable of accepting Boolean equation data entry, they offer other input alternatives, such as gate-level graphics and higher-level state machine descriptive language, and allow features such as set-definition and automated logic reduction. Several higher-level packages designed for use with all PLDs are generally available. Two of the most popular are CUPL® software from Logical Devices and ABEL® software from Data I/O.

For more detailed information see Section 4.

National Semiconductor

Programmable Logic Design Methodology

PLDs offer a number of design advantages to the designer. But in order to make use of these, the designer must apply a specific design methodology in order to maximize the effectiveness of the PLD tools available. This chapter outlines the steps by which this is done and illustrates their use by specific examples. Details of the use of the most important tools are given in Section 4. For more diverse examples without as much emphasis on methodology, refer to Applications examples Sections 2 and 3.

Design Development Process

The design development process for PLDs proceeds in three main phases:

- Logic design
- Design implementation
- Design/Logic verification

Within each phase, an experienced designer will pass through a number of steps. While the process may appear involved at first, it is mostly a stylization of good design practices and efficient use of the PLD tools available to the designer.

Within the Logic Design phase, the steps involved are common procedure for any digital design engineer and can be tailored to suit the individual taste and design requirements. The main steps are:

- Define the system problem
- Generate a block diagram
- Implement the function logically
- Derive the Boolean equations describing the design

These are largely self-explanatory. For readers requiring some background information on logic design principles, refer to Appendix A.

Design Implementation consists largely of selecting and using the tools to translate the results of the first phase into a configure d PLD. It consists of steps:

- PLD family and device selection
- Partitioning the logic to fit the devices selected
- Equation entry
- Running development software and JEDEC file creation
- Platform and programmer configuration
- JEDEC file transfer
- Device programming

The preparation of software, platform and programmer need be done ony for the initial use of PLDs. Following that, the other functions are all straightforward operations often handled automatically by the PLD tools selected and not requiring involvement on the part of the designer.

Design verification is the final phase during which the correct programming of the device is checked, along with the generation of test procedures which verify that the device itself implements what was originally required. The steps in this phase are:

- Device programming verification
- Design test vector generation
- Device simulation
- Device functional test
- Design documentation

The effort involved in each depends both on the design complexity and the tools available. As with any other design, the verification phase can be too easily overlooked in the entire design process, but effort spent in judicious testing and adequate documentation is normally well spent.

Each of these steps is described in detail later in this section.

Logic Design

This section gives a detailed account of the steps involved in generating the initial theoretical design, illustrated by reference to an example of a 6-bit bidirectional shift register.

DEFINING THE PROBLEM

As with any other design methodology, the first step involved is a clear definition of the problem to be solved. In the case of the shift device, what is required is a device with the following characteristics:

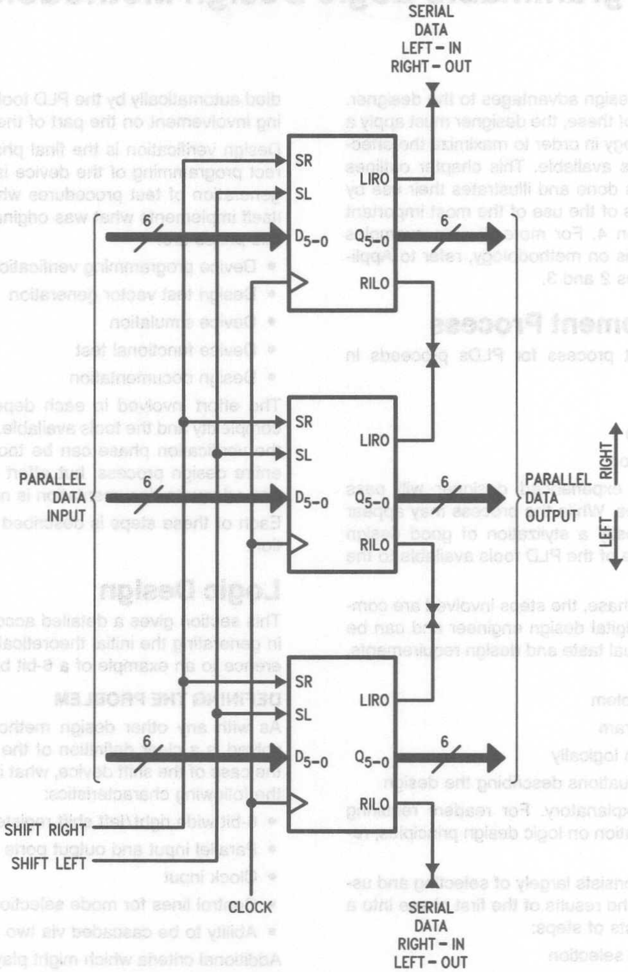
- 6-bit wide right/left shift register
- Parallel input and output ports
- Clock input
- Control lines for mode selection
- Ability to be cascaded via two bidirectional serial ports

Additional criteria which might play a role in selection of the final solution are the need for low parts count, power and speed considerations, and the need to interface with or mop up other logic in the area. For the purpose of this example, assume these criteria impose no special constraints.

DESIGNING THE LOGIC

Based on the above criteria, the block diagram of the logic can be generated directly, as shown in *Figure 1*. The signal names are given to permit unambiguous reference to their function and any considerations of logic context within the system should be incorporated here.

From the block diagram, the designer derives the detailed functional description of the intended behavioral concept. This may take a number of forms, depending on the application and the preference of the designer. One common method of expressing the detailed operation of a registered application such as this is a function table, which is shown in *Figure 2* for reference. Another common method is the use of timing waveforms which are omitted for this example due to its simplicity. The target function is further defined by deriving a detailed logic schematic (as shown in *Figure 3*), combinatorial truth table, or direct expression in Boolean equations.



TL/L/9988-1

FIGURE 1. Block Diagram Showing 3 Cascaded Shift Registers

Function	SL	SR	RILO	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	LIRO
Hold	0	0	Z	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	Z
Shift Right	0	1	RI	RI	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
Shift Left	1	0	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	LI	LI
Parallel Load	1	1	Z	D_5	D_4	D_3	D_2	D_1	D_0	Z

FIGURE 2. Functional Operation Table for Shifter Example

DERIVING BOOLEAN EQUATIONS

In order to provide a definition of the circuit which the design tools can handle, it is usually necessary to express the design in terms of Boolean equations. The fundamental transfer function of a PAL® device is the sum-of-products or, through DeMorgan inversion, product-of-sums form. Logic equations can be derived directly from the function table shown in *Figure 2*, the logic schematic in *Figure 3*, or from the method of logic implementation preferred.

For any TRI-STATE output, including bidirectional I/O lines, additional equations may need to be specified to define the control functions of these lines. This is illustrated in the example of the 6-bit shift register.

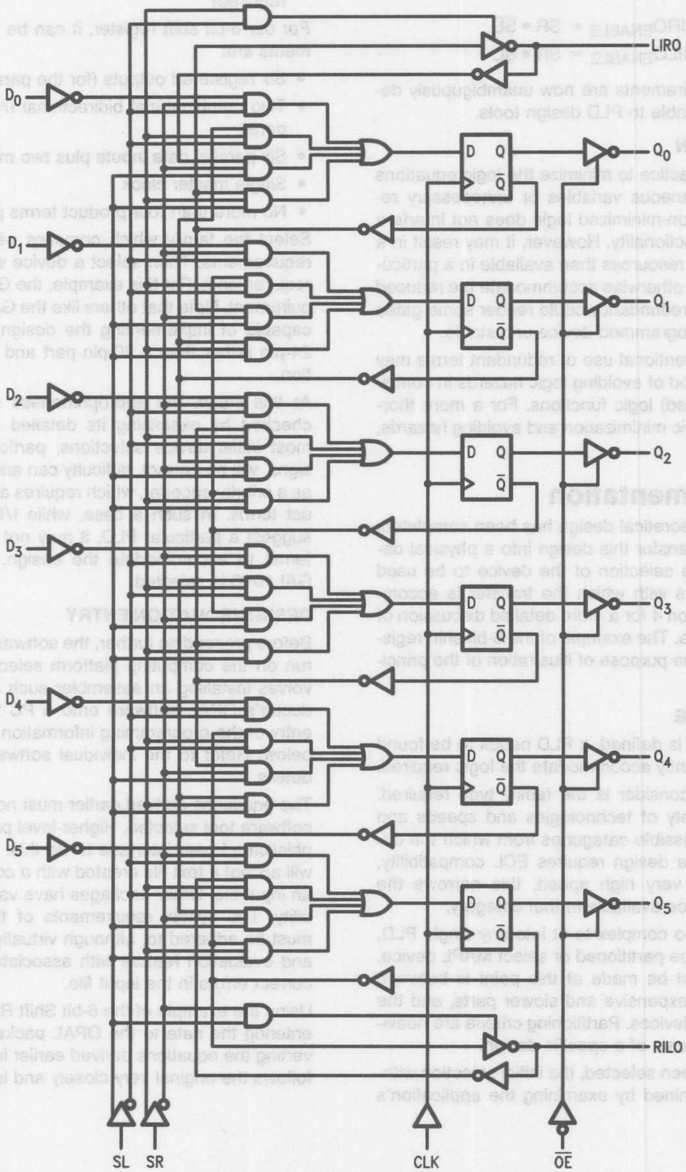


FIGURE 3. Gate-Level Logic Schematic of Shifter

$+ D_i \bullet SR \bullet SL, i = 1 \ldots 4$

$Q_5 \leftarrow Q_5 \bullet \overline{SR} \bullet \overline{SL} + RILO \bullet SR \bullet \overline{SL} + Q_4 \bullet \overline{SR} \bullet SL$
$+ D_5 \bullet SR \bullet SL$

$LIRO_{OUTPUT} = Q_0$; $LIRO_{ENABLE} = SR \bullet \overline{SL}$

$RILO_{OUTPUT} = Q_5$; $RILO_{ENABLE} = \overline{SR} \bullet SL$

The device logic requirements are now unambiguously defined in a form acceptable to PLD design tools.

LOGIC MINIMIZATION

It is generally good practice to minimize the logic equations to eliminate any extraneous variables or unnecessary redundant min-terms. Non-minimized logic does not interfere with proper device functionality. However, it may result in a design requiring more resources than available in a particular device which could otherwise accommodate the reduced equations. Also, logic redundancy could render some gates or nodes within the programmed device untestable.

On the other hand, intentional use of redundant terms may be a convenient method of avoiding logic hazards in combinatorial (unsynchronized) logic functions. For a more thorough discussion of logic minimization and avoiding hazards, refer to Appendix A.

Design Implementation

Now that the basic theoretical design has been completed, the next stage is to transfer this design into a physical device. This requires the selection of the device to be used and a number of tools with which the transfer is accomplished. Refer to Section 4 for a more detailed discussion of PLD tools and their use. The example of the 6-bit shift register is again used for the purpose of illustration of the principles.

SELECTING A DEVICE

Once the logic design is defined, a PLD needs to be found which can most efficiently accommodate the logic required.

The first criterion to consider is the family type required. PLDs come in a variety of technologies and speeds and offer a spectrum of possible categories from which the device is selected. If the design requires ECL compatibility, CMOS low-power or very high speed, this narrows the choice down to a device available in that category.

If the target logic is too complex to fit into any single PLD, then the design must be partitioned or select MAPL device. A decision which must be made at this point is between using more complex, expensive and slower parts, and the more traditional GAL devices. Partitioning criteria are heavily dependent on the goals of a specific design.

Once the family has been selected, the initial selection within the family is determined by examining the application's

- Clocking requirements
- Complexity of each logic equation (number of min-terms required)

For our 6-bit shift register, it can be seen that the requirements are:

- Six registered outputs (for the parallel-out lines)
- Two combinatorial bidirectional I/O lines (for the serial ports)
- Six parallel data inputs plus two mode control inputs
- Single master clock
- No more than four product terms per output function

Select the family which complies with the overall system requirements. Then select a device which furnishes all the requirements. For this example, the GAL16V8 fulfills the requirement. Note that others like the GAL20V8 would also be capable of implementing the design, but would involve a 24-pin rather than a 20-pin part and higher power dissipation.

At this stage, the appropriateness of the device is also checked by examining its detailed block diagram. While most initial device selections, particularly for simpler designs, will be correct, difficulty can arise with a design, such as a priority encoder, which requires a large number of product terms. In such a case, while I/O requirements might suggest a particular PLD, it may not offer enough product terms to accommodate the design. For this design, the GAL16V8 is selected.

DESIGN/EQUATION ENTRY

Before proceeding further, the software tools will need to be run on the computing platform selected. This normally involves installing an assembler such as National Semiconductor's OPAL software onto a PC in preparation for the entry of the programming information in the form described below. Refer to the individual software documentation for details.

The equations derived earlier must now be entered into the software tool selected. Higher-level packages provide a sophisticated user interface to do this. The OPAL assembler will accept a text file created with a common editor utility as an input file. Other packages have varying degrees of flexibility. The syntax requirements of the software package must be adhered to, although virtually all provide a parsing and evaluation feature with associated error messages to correct errors in the input file.

Using the example of the 6-bit Shift Register, the method of entering the data to the OPAL package involves first converting the equations derived earlier into OPAL syntax. This follows the original very closely and is shown in *Figure 4*.

```
; pin 1   2    3  4  5  6  7  8  9     10
      CLK SR   D0 D1 D2 D3 D4 D5 SL    GND
; pin 11  12   13 14 15 16 17 18 19    20
      /G  RILO Q5 Q4 Q3 Q2 Q1 Q0 LIRO  VCC

equations

      /Q0 := /Q0 * /SR * /SL
           + /Q1 *  SR * /SL
           + /LIRO * /SR * SL
           + /D0 *  SR *  SL
      /Q1 := /Q1 * /SR * /SL
           + /Q2 *  SR * /SL
           + /Q0 * /SR * SL
           + /D1 *  SR *  SL
      /Q2 := /Q2 * /SR * /SL
           + /Q3 *  SR * /SL
           + /Q1 * /SR * SL
           + /D2 *  SR *  SL
      /Q3 := /Q3 * /SR * /SL
           + /Q4 *  SR * /SL
           + /Q2 * /SR * SL
           + /D3 *  SR *  SL
      /Q4 := /Q4 * /SR * /SL
           + /Q5 *  SR * /SL
           + /Q3 * /SR * SL
           + /D4 *  SR *  SL
      /Q5 := /Q5 * /SR * /SL
           + /RILO *  SR * /SL
           + /Q4 * /SR * SL
           + /D5 *  SR *  SL
      /LIRO = /Q0
      LIRO.OE = SR * /SL
      /RILO = /Q5
      RILO.OE = /SR * SL
```

TL/L/9988-4

FIGURE 4. OPAL™ Equation File for Shifter Example

```
Log file for 6shft.eqn
Device: 16V8

Pin   Label              Type
---   -----              ----
1     CLK                clock pin
2     SR                 pos,com input
3     D0                 pos,com input
4     D1                 pos,com input
5     D2                 pos,com input
6     D3                 pos,com input
7     D4                 pos,com input
8     D5                 pos,com input
9     SL                 pos,com input
10    GND                ground pin
11    /G                 enable pin
12    RILO               neg,trst,com feedback(bidir)
13    Q5                 neg,reg feedback
14    Q4                 neg,reg feedback
15    Q3                 neg,reg feedback
16    Q2                 neg,reg feedback
17    Q1                 neg,reg feedback
18    Q0                 neg,reg feedback
19    LIRO               neg,trst,com feedback(bidir)
20    VCC                power pin
```

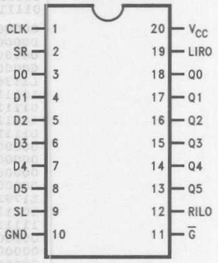
TL/L/9988-8

```
Device Utilization:

No of dedicated inputs used              :  8/8   (100.0%)
No of feedbacks used                     :  8/8   (100.0%)

           ----------------------------------------
           Pin   Label                Terms Usage
           ----------------------------------------
           19    LIRO.oe               1/1   (100.0%)
           18    Q0                    4/8   (50.0%)
           17    Q1                    4/8   (50.0%)
           16    Q2                    4/8   (50.0%)
           15    Q3                    4/8   (50.0%)
           14    Q4                    4/8   (50.0%)
           13    Q5                    4/8   (50.0%)
           12    RILO.oe               1/1   (100.0%)
           ----------------------------------------
           Total Terms                28/64  (43.8%)
           ----------------------------------------
```

TL/L/9988-10

FIGURE 6

Chip Diagram (DIP)



TL/L/9988-9

```
          title    6-bit cascadable shift register
          pattern  6shft
          revision C
          author   Tarif Arabi
          company  National Semiconductor Corporation
          Date     8/30/92

          *
NOTE PINS CLK:1 SR:2 D0:3 D1:4 D2:5 D3:6 D4:7 D5:8 SL:9 GND:10*
NOTE PINS /G:11 RILO:12 Q5:13 Q4:14 Q3:15 Q2:16 Q1:17 Q0:18 LIRO:19*
NOTE PINS VCC:20*
QF2194*QP20*F0*
L0000
01111111111111111111111111111011
11111101111111111111111111111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L0256
10111110111111111111111111111011
01111111110111111111111111111011
10101111111111111111111111110111
01111011111111111111111111110111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L0512
10111111111011111111111111111011
01111111111111101111111111111011
10111110111111111111111111110111
01111111101111111111111111110111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L0768
10111111111111101111111111111011
01111111111111111110111111111011
10111111111011111111111111110111
01111111111110111111111111110111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L1024
10111111111111111110111111111011
01111111111111111111111011111011
10111111111111101111111111110111
01111111111111111011111111110111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L1280
10111111111111111111111011111011
01111111111111111111111111101011
10111111111111111110111111110111
01111111111111111111101111110111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L1536
10111111111111111111111111101011
01111111111111111111111111111010
10111111111111111111111011110111
01111111111111111111111110110111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L1792
10111111111111111111111111110111
11111111111111111111111111101111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L2048
00000000*
L2056
0000000000000000000000000000000000000000000000000000000000000000*
L2120
10000001*
L2128
1100000011110000111100001111000011110000111100001111000011000000*
L2192
01*
C6826*
♥0000
0111100001111000011000000*
L2192
01*
C6826*
♥0000
```

FIGURE 5. JEDEC File for Shifter Example Produced by OPAL Software

COMPILATION—CREATING THE JEDEC FILE

At this stage, pin assignment is normally made, either manually or automatically by the software. OPAL offers an automatic assignment, which can then be edited manually, if desired.

Once the equation file has been entered and pin assignments resolved, the assembly is performed on the platform, the results of which are a JEDEC fuse map for down-loading to the programmer.

In order for the equations to be converted into a bit pattern from which the cells of a PLD can be systematically programmed, the initial equations must be converted into a form, known as the JEDEC file. The JEDEC file is an industry-wide standard accepted by all programming hardware. It consists of a formatted table indicating all of the cells (fuses) in the PLD to be programmed to implement the specified logic functions. This is done by a module within the software tool known as the assembler. Details of operation of the assembler varies from one package to another, but each provides syntax checking and an evaluation of whether the design can be implemented in the device chosen, as well as the JEDEC file itself, which is normally stored on disk ready for down-loading to a programmer.

The actual form of the JEDEC file is usually of little interest to the system designer. Its only purpose is to provide a uniform interface between commercial PLD software and hardware tools and no real information for the designer is provided by its details. It may occasionally be useful as a debugging aid to isolate any problems occurring between equation entry and functional test.

PROGRAMMING HARDWARE PREPARATION

Before the cell data can be transferred, the programmer needs to be connected to the software platform and fitted with any socket adapters required to accommodate the blank sample device.

For the purpose of this example, a GAL16V8 is being programmed with a Data I/O Model 29 programmer equipped with Logic-Pack. In order to do this, connect the System 29 to the platform via an RS232C cable, according to the system documentation. Look up the GAL16V8 on the device chart to determine and enter the family and pinout code.

DEVICE PROGRAMMING

This step involves transferring the prepared JEDEC file across a communication link from the platform to the hardware programmer. Each programmer differs slightly, but generally each requires that a number of prompts be answered with such information as file name, device type and manufacturer. Usually, a test is run at this time on the device by the programmer which ensures that the device is correctly oriented in the socket and is in fact blank and able to be programmed. The correctness of data transfer is verified by means of a checksum transmitted with the file.

For the purposes of our example, the System 29 provides all the prompts required to do this.

Now that all relevant information has been entered into the programmer, it is a matter of simply invoking the Program function.

This translates the JEDEC file into addresses, data patterns and programming pulses, which, when applied to the pins of the device in the socket, will configure the cells of the device in a pattern which will cause the device to operate in accordance with the original design. The implementation of the design in the PLD has now been completed.

Again, for the purposes of our example, the System 29 provides all the prompts required to do this.

Logic Verification

Verification is required to ensure not only that the device has been configure d exactly as intended, but also that the programmer has functioned correctly and that the design performs as originally intended. Again, this takes the form of several steps.

PATTERN VERIFICATION

This may be performed automatically by the programmer. If not, it is recommended that a manual verification run is performed while the device is still in the programmer to ensure that the pattern set in the device corresponds to that specified by the fuse map in the JEDEC file. This is a simple step which is done by the programmer itself. The programmer reads the pattern directly from the PLD, similar to reading a PROM, and compares this directly with the original JEDEC file still resident in the programmer.

TEST VECTOR GENERATION

Particularly with more complex designs, it is recommended that some consideration for a set of device exercises, generally known as test vectors, be given as early as the equation entry stage above. Some advanced software tools may provide automatic generation of test vectors from the equations as they are entered. Otherwise, vectors must be generated by hand. Even in the case of automatic test vector generation, some designers prefer to add their own additional vectors to verify application-specific operations.

For a design of the complexity of our 6-bit shift register, test vectors are easily generated by hand, as shown in *Figure 7.* In this case, a test such as walking ones and zeroes is probably sufficient to prove functionality of the device beyond reasonable doubt. For more complex designs, it may be helpful to employ fault-grading software to ensure adequate coverage of all design paths and gates by the test vectors.

DESIGN SIMULATION

This optional step generates the device output vectors which allow verification of correct design operation. This can be done manually, or with the help of the simulator module of the software tool like OPAL software to predict the output configuration for the device based on the original software model entered as Boolean equations.

The output vectors from the simulation must be examined to confirm that the model operates correctly. They also provide the output states, which must accompany the test vectors for functional testing of the device.

Beyond device-level simulation, additional software is becoming available to generate models of the programmed PLDs for use in system-level simulations. Such simulations are typically performed on CAD workstations and, more recently, personal computers.

DEVICE FUNCTIONAL TESTING

The device is evaluated fully for correct performance of the function desired. In the case of the 6-bit shift register, this would involve checking all the functions outlined in the function table in *Figure 2.*

Varying from one software tool to another, the test vectors are entered (if not generated automatically) into the soft ware, which appends them in the proper format to the JEDEC file, as shown in *Figure 8.* The test vector entry/generation typically follows the equation entry step, so that the combined JEDEC file is down-loaded to the programmer. Later, following device programming and pattern verification steps, the programmer performs the functional test on the PLD while still in the socket. The input vector waveforms are applied to the device pins while in the normal operational mode, and the device output signals are compared with the expected output vectors.

As a final step, most PLDs include a "security cell/fuse" which, when programmed, disables further programming and verifying. This prevents direct copying of the logic patterns resulting in proprietary custom circuits that are difficult to copy or reverse engineer.

Inputs								Bidirectional I/O		Outputs						Comments
SL	SR	D_5	D_4	D_3	D_2	D_1	D_0	LIR0	RIL0	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	
1	1	0	0	0	0	0	0	1	1	L	L	L	L	L	L	Load all zeroes.
0	0	1	1	1	1	1	1	1	1	L	L	L	L	L	L	Hold zeroes.
1	0	1	1	1	1	1	1	1	L	L	L	L	L	L	H	Shift left single one,
1	0	1	1	1	1	1	1	0	L	L	L	L	L	H	L	followed by zeroes.
1	0	1	1	1	1	1	1	0	L	L	L	L	H	L	L	
1	0	1	1	1	1	1	1	0	L	L	L	H	L	L	L	
1	0	1	1	1	1	1	1	0	L	L	H	L	L	L	L	One shifts out of RIL0,
1	0	1	1	1	1	1	1	0	H	H	L	L	L	L	L	and vanishes.
1	0	1	1	1	1	1	1	0	L	L	L	L	L	L	L	Load all ones.
1	1	1	1	1	1	1	1	0	0	H	H	H	H	H	H	Hold ones.
0	0	0	0	0	0	0	0	0	0	H	H	H	H	H	H	Shift right single zero,
0	1	0	0	0	0	0	0	H	0	L	H	H	H	H	H	followed by ones.
0	1	0	0	0	0	0	0	H	1	H	L	H	H	H	H	
0	1	0	0	0	0	0	0	H	1	H	H	L	H	H	H	
0	1	0	0	0	0	0	0	H	1	H	H	H	L	H	H	
0	1	0	0	0	0	0	0	H	1	H	H	H	H	L	H	
0	1	0	0	0	0	0	0	L	1	H	H	H	H	H	L	Zero shifts out of LIR0,
0	1	0	0	0	0	0	0	H	1	H	H	H	H	H	H	and vanishes.

Key: 0 = Apply Low Input, 1 = Apply High Input
L = Expect Low Output, H = Expect High Output

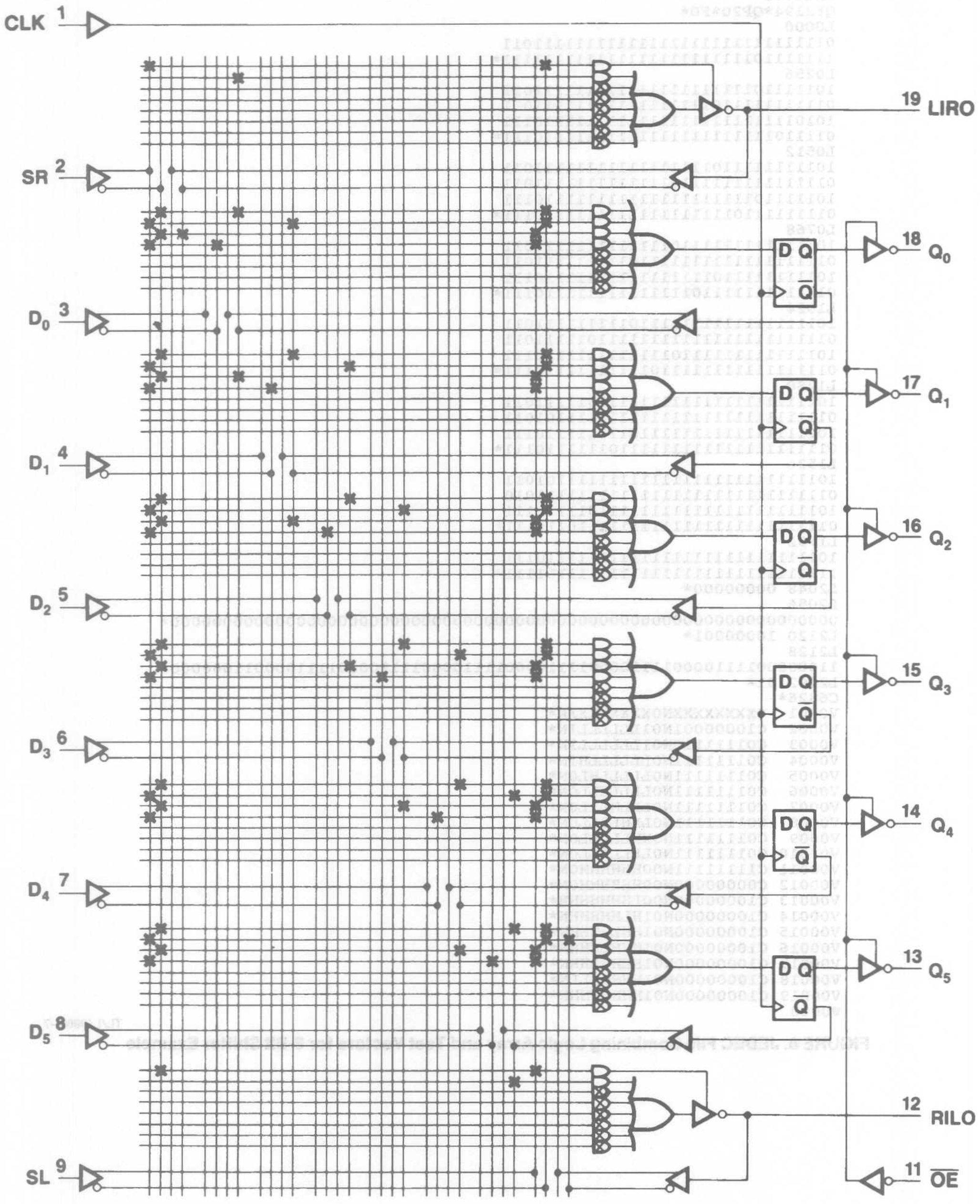
Note: The device is clocked after applyling each input vector. Outputs on the same line are strobed and compared after the clock.

FIGURE 7. Functional Test Pattern for 6-Bit Shift Register Example

```
10111110111111111111111111111011
01111111111011111111111111111011
10101111111111111111111111110111
01111011111111111111111111110111*
L0512
10111111111011111111111111111011
01111111111111101111111111111011
10111110111111111111111111110111
01111111101111111111111111110111*
L0768
10111111111111101111111111111011
01111111111111111110111111111011
10111111111011111111111111110111
01111111111110111111111111110111*
L1024
10111111111111111110111111111011
01111111111111111111111011111011
10111111111111101111111111110111
01111111111111111011111111110111*
L1280
10111111111111111111111011111011
01111111111111111111111111101011
10111111111111111110111111110111
01111111111111111111101111110111*
L1536
10111111111111111111111111101011
01111111111111111111111111111010
10111111111111111111111011110111
01111111111111111111111110110111*
L1792
10111111111111111111111111110111
11111111111111111111111111101111*
L2048 00000000*
L2056
0000000000000000000000000000000000000000000000000000000000000000*
L2120 10000001*
L2128
1100000011110000111100001111000011110000111100001111000011000000*
L2192 01*
C6826*
V0001  0XXXXXXXXN0XXXXXXXXN*
V0002  C10000001N01LLLLLL1N*
V0003  C01111110N01LLLLLL1N*
V0004  C01111111N0LLLLLLH1N*
V0005  C01111111N0LLLLLHL0N*
V0006  C01111111N0LLLLHLL0N*
V0007  C01111111N0LLLHLLL0N*
V0008  C01111111N0LLHLLLL0N*
V0009  C01111111N0HHLLLLL0N*
V00010 C01111111N0LLLLLLL0N*
V00011 C11111111N00HHHHHH0N*
V00012 C00000000N00HHHHHH0N*
V00013 C10000000N00LHHHHHHN*
V00014 C10000000N01HLHHHHHN*
V00015 C10000000N01HHLHHHHN*
V00016 C10000000N01HHHLHHHN*
V00017 C10000000N01HHHHLHHN*
V00018 C10000000N01HHHHHLLN*
V00019 C10000000N01HHHHHHHN*
♥0000
```

TL/L/9988-7

FIGURE 8. JEDEC File Combining Logic Array and Test Vectors for 6-Bit Shifter Example



TL/L/9988-3

FIGURE 9. GAL16V8 Logic Diagram Showing Fuse Pattern of Shifter Example

National Semiconductor

Fabrication of Programmable Logic

PLD Technologies

National Semiconductor is a broad-based supplier of programmable logic products. PLD's are offered in a wide range of circuit and programming technologies to address the diverse needs of most customer applications. PLD's can be seen as a stage of sophistication in the continuum from standard TTL logic to full-custom circuits. To provide the flexibility of configurable logic, PLD's make use of a number of circuit and programming technologies. These vary, depending on the type of PLD.

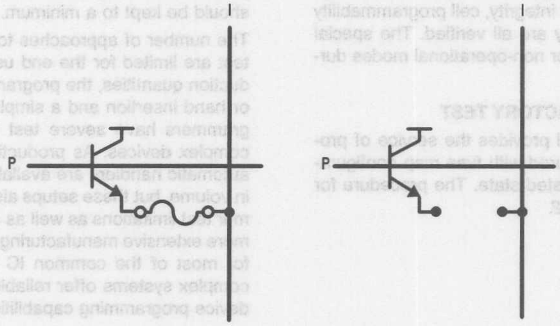
ECL PAL DEVICES

For ECL systems, high-speed PAL devices have been implemented with ECL I/Os using an oxide-isolated process known as OXISS. OXISS is a fully ion-implanted Schottky bipolar process with a 2 micron minimum feature size and two-layer metal interconnect. The ECL PAL products use the same Titanium-Tungsten lateral fuses as used in the original standard, Series-A and Series-B TTL PAL devices. The ECL PAL products are also available with both 10 KH and 100K compatibility.

More recent developments are based on National's ASPECT (Advanced Single-Poly Emitter-Coupled Technology) process. This is an oxide-isolated, self-aligned, contactless poly-emitter process which uses a single poly and two metal interconnect layers. The smaller geometries result in both lower gate power requirements and a higher device speed.

EECMOS TECHNOLOGY

Electrically-Erasable CMOS offers many advantages as a technology for PLDs. Most importantly, it offers the ability to erase and reprogram devices. This allows lower part usage, particularly at the development and prototype stages, by allowing the same device to be re-used or revised. This extends into manufacturing because the technology allows 100% factory testability without encountering the cycling difficulties or windowed packages associated with UV-based devices. 100% programming and functional yields are possible. Under this technology GAL and MAPL devices are manufactured.



Unprogrammed

Programmed

TL/L/9990-1

FIGURE 1. Lateral Fuse Circuit

The power requirements of such a technology are very low when compared to standard bipolar. The technology has developed to a point where performance is comparable to standard PLDs, if not to the higher-speed parts. The low-power characteristics permit higher circuit density and higher reliability, which are particularly important in remote or power-conscious environments, such as in telecommunications.

The EECMOS technology is under further development to allow such devices to be programmed while in circuit, which will add even more to their flexibility and usefulness in manufacturing since they can be assembled in-circuit with all other components before being configured.

Quality and Factory Testing

PRODUCT RELIABILITY

National Semiconductor implements a reliability program for all of its integrated circuits. Reliability data is available for individual parts from the local sales office. It consists of:

- New product, package and process qualifications
- Existing product, package and process change qualifications
- Existing product, package and process monitoring

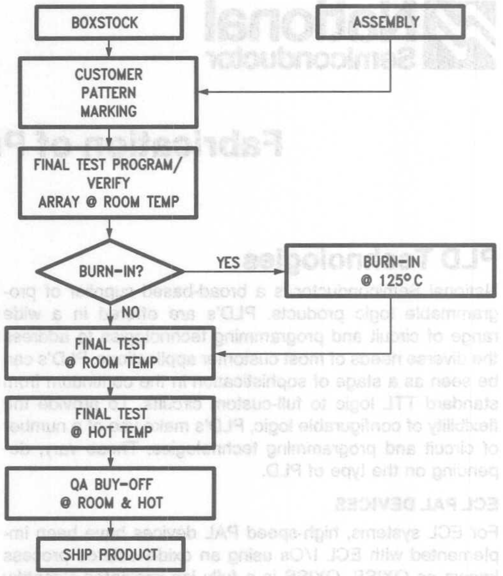
Product qualification testing performed by National meets or exceeds MIL standard 883. For a more thorough discussion of product quality and reliability, refer to the National Semiconductor Reliability Handbook.

TEST CIRCUITRY

All ECL PAL devices from National Semiconductor are fabricated with a number of test fuses and dedicated test circuitry as a part of the device. During final testing of each device, the functional paths, electrical integrity, cell programmability and the programming circuitry are all verified. The special test circuitry is accessed under non-operational modes during functional testing.

PRE-PROGRAMMED PLD FACTORY TEST

As discussed earlier, National provides the service of providing devices already configured with fuse map configuration to customers in a fully-tested state. The procedure for doing this is shown in *Figure 2*.



TL/L/9990–7

FIGURE 2. Pre-Programmed PLD Final Test Flow

CUSTOMER HANDLING AND TEST

Care must be taken to prevent static buildup when handling devices, particularly CMOS devices. Handling is the primary cause of device failures, and to minimize problems during production, the number of steps requiring device handling should be kept to a minimum.

The number of approaches to reliable handling and device test are limited for the end user of PLDs. For smaller production quantities, the programming hardware may be used or hand insertion and a simple functional test, though programmers have severe test limitations for high-speed or complex devices. As production quantities become larger, automatic handlers are available which can handle devices in volume, but these setups also are subject to the programmer test limitations as well as other protential problems. For more extensive manufacturing flows, programs are available for most of the common IC tester systems. These more complex systems offer reliable test capabilities, as well as device programming capabilities.

Section 2

Low Density GAL and PAL Devices

Section 2 Contents

GAL AND PAL DATASHEETS

GAL16V8/A 20-Pin Generic Array Logic Family 2-3
GAL20V8/A 24-Pin Generic Array Logic Family 2-22
GAL16V8QS 20-Pin Generic Array Logic Family 2-40
GAL20V8QS 24-Pin Generic Array Logic Family 2-59
GAL22V10 Generic Array Logic 2-78
GAL20RA10 Generic Array Logic 2-91
GAL6001 Generic Array Logic 2-102
ECL PAL10/10016P8 Combinatorial PAL 2-113
ECL PAL10/10016P8-3 3 ns Combinatorial PAL 2-122
ECL PAL10/10016PE8-3 3 ns Combinatorial PAL 2-127
ECL PAL10/10016P4A Combinatorial PAL 2-132
ECL PAL10/10016P4-2 2 ns Combinatorial PAL 2-136
ECL PAL10/10016C4-2 2 ns Combinatorial PAL 2-141

GAL AND PAL APPLICATION EXAMPLES

1.0 Basic Gates 2-146
2.0 Basic Flip-Flops 2-151
3.0 Quad 4-to-1 Multiplexer 2-156
4.0 Dual 8-to-1 Multiplexer 2-160
5.0 7-Bit Counter with Parallel-Load 2-165
6.0 10-Bit Up/Down Counter 2-169
7.0 8-Bit Barrel Shifter 2-177
8.0 Hexadecimal 7-Segment Display Encoder 2-181
9.0 Dual-Port RAM Controller 2-187
10.0 CPU Board Random Control Logic 2-196

GAL AND PAL APPLICATION NOTES

AN-707 High Speed System Design Using Programmable GAL Devices 2-198
AN-799 PAL to GAL Conversion 2-209
AN-667 A GAL6001-30L Zero Wait State Page Mode Memory System Interface Between the DP8422A and the 68020 2-220
AN-669 A GAL Interface Between Static Random Access Memory (SRAM) and the NSC Raster Graphics Processor (RGP, DP8500) 2-230
AN-618 A GAL Interface for a 25 MHz and above No-Wait State DP8422A/80286 Burst Mode DRAM Memory System 2-237
AN-535 A GAL Interface for a Dual Access DP8422A/68030/74F632 Error Detecting and Correcting Memory System 2-246

National Semiconductor

GAL16V8/A 20-Pin Generic Array Logic Family

General Description

The EECMOS GAL® 16V8/A devices are fabricated using electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 20-pin GAL16V8 features 8 programmable Output Logic Macrocells (OLMCs) allowing each TRI-STATE® output to be configured by the user. Additionally, the GAL16V8 is capable of emulating, in a functional/fuse map/parametric compatible device, all common 20-pin PAL® device architectures.

Programming is accomplished using readily available hardware and software tools. NSC guarantees a minimum 100 erase/write cycles.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, NSC guarantees 100% field programmability and functionality of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

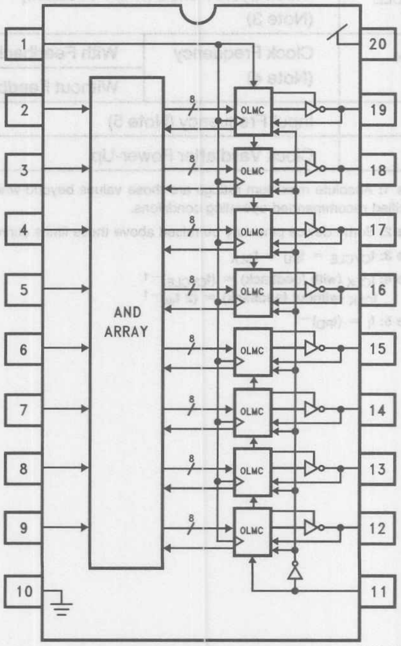
Features

- High performance EECMOS technology
  - — 7.5 ns maximum propagation delay
  - — f_{CLK} = 100 MHz
  - — 5 ns maximum from clock input to data output
  - — TTL compatible 24 mA outputs
- Reduced power
  - — Low power = 115 mA I_{CC} max, 75 mA Typ
- Electrically erasable cell technology
  - — Reconfigurable logic
  - — Reprogrammable cells
  - — 100% tested/guaranteed 100% yields
  - — High speed electrical erasure (<50 ms)
  - — 20 year data retention
- Eight output logic macrocells
  - — Maximum flexibility for complex logic designs
  - — Programmable output polarity
  - — Also emulates 20-pin PAL devices with full function/fuse map/parametric compatibility
- Preload and power-up reset of all registers
  - — 100% functional testability
- Fully supported by National OPAL™ and OPALjr development software
- Security cell prevents copying logic
- Electronic signature for identification

PAL Replacement by Device Type

"Small PAL" Mode				"Registered PAL" Mode			"Medium PAL" Mode
10L8	12L6	14L4	16L2	16R8	16R6	16R4	16L8
10H8	12H6	14H4	16H2	16RP8	16RP6	16RP4	16H8
10P8	12P6	14P4	16P2				16P8

Block Diagram—GAL16V8



TL/L/11255-1

GAL16V8-7/-10

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage V_{CC}	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	> 2000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-026 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Min	Nom	Max	Units
V_{CC}	Supply Voltage	4.75	5	5.25	V
T_A	Operating Free-Air Temperature	0	25	75	°C
T_C	Operating Case Temperature				°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL16V8-7L COM		GAL16V8-10L COM		Units
			Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		7		10		ns
t_H	Hold Time (Input after Clock)		0		0		ns
t_W	Clock Pulse Width (High/Low)		5		8		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		12		17		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		83.3		58.8	MHz
		Without Feedback		100		62.5	
f_I	Input Frequency (Note 5)			133.33		100	
t_{PR}	Clock Valid after Power-Up			100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) $= (t_{CYCLE})^{-1}$
f_{CLK} (without feedback) $= (2\ t_W)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		$V_{CC}+1$	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min I_{OH} = 3.2 mA	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min I_{OL} = 24 mA			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC}(Max)			10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND			−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC}(Max)			10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC}(Max)			10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND			−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND	−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max		75	115	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V		5	8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V		5	8	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL16V8-7		GAL16V8-10		Units
			COM		COM		
			Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, C_L = 50 pF		7.5		10	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, C_L = 50 pF		5		7	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High: S1 Open, C_L = 50 pF Active Low: S1 Closed, C_L = 50 pF		6		10	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}: S1 Open, C_L = 5 pF From V_{OL}: S1 Closed, C_L = 5 pF		6		10	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High: S1 Open, C_L = 50 pF Active Low: S1 Closed, C_L = 50 pF		9		10	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}: S1 Open, C_L = 5 pF From V_{OL}: S1 Closed, C_L = 5 pF		9		10	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, C_L = 50 pF		45		45	μs

GAL16V8A-12/-15

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage V_{CC}	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-026 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C
T_C	Operating Case Temperature							°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL16V8A-12L COM		GAL16V8A-15L COM/IND		Units
			Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		10		12		ns
t_H	Hold Time (Input after Clock)		0		0		ns
t_W	Clock Pulse Width (High/Low)		8		8		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		20		22		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		50		45.5	MHz
		Without Feedback		62.5		62.5	
f_I	Input Frequency (Note 5)			83.3		66.6	
t_{PR}	Clock Valid after Power-Up			100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) = $(t_{CYCLE})^{-1}$
f_{CLK} (without feedback) = $(2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL16V8A-12/-15 (Continued)

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage			2.0		$V_{CC} + 1$	V
V_{IL}	Low Level Input Voltage			−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, I_{OH} = −3.2 mA		2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 24 mA				0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC}(Max)				10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND				−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC}(Max)				10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC}(Max)				10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND				−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND		−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	COM			90	mA
			IND			130	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V				8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V				10	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL16V8A-12L		GAL16V8A-15L		Units
			COM		COM/IND		
			Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, C_L = 50 pF		12		15	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, C_L = 50 pF		10		10	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High: S1 Open, C_L = 50 pF Active Low: S1 Closed, C_L = 50 pF		10		15	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}: S1 Open, C_L = 5 pF From V_{OL}: S1 Closed, C_L = 5 pF		10		15	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High: S1 Open, C_L = 50 pF Active Low: S1 Closed, C_L = 50 pF		12		15	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}: S1 Open, C_L = 5 pF From V_{OL}: S1 Closed, C_L = 5 pF		12		15	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, C_L = 50 pF		45		45	μs

GAL16V8-20/-25

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage V_{CC}	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C

Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-026 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C
T_C	Operating Case Temperature							°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL16V8-20L		GAL16V8-25L		GAL16V8-25Q		Units
			COM		COM		COM		
			Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		15		15		15		ns
t_H	Hold Time (Input after Clock)		0		0		0		ns
t_W	Clock Pulse Width (High/Low)		12		12		12		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		27		27		27		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		37		37		37	MHz
		Without Feedback		41.66		41.66		41.66	
f_I	Input Frequency (Note 5)			50		50		50	
t_{PR}	Clock Valid after Power-Up			100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) $= (t_{CYCLE})^{-1}$
f_{CLK} (without feedback) $= (2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL16V8-20/-25 (Continued)

Electrical Characteristics Over Recommended Operating Conditions

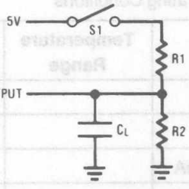
Symbol	Parameter	Conditions	Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage			2.0		$V_{CC}+1$	V
V_{IL}	Low Level Input Voltage			−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, I_{OH} = −3.2 mA		2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 24 mA				0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC}(Max)				10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND				−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC}(Max)				10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC}(Max)				10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND				−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND		−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	-20L, -25L			90	mA
			-25Q			55	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V				8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V				8	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL16V8-20L		GAL16V8-25L		GAL16V8-25Q		Units
			COM		COM		COM		
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, C_L = 50 pF		20		25		25	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, C_L = 50 pF		12		12		12	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High: S1 Open, C_L = 50 pF Active Low: S1 Closed, C_L = 50 pF		18		20		20	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}: S1 Open, C_L = 5 pF From V_{OL}: S1 Closed, C_L = 5 pF		18		20		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High: S1 Open, C_L = 50 pF Active Low: S1 Closed, C_L = 50 pF		20		25		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}: S1 Open, C_L = 5 pF From V_{OL}: S1 Closed, C_L = 5 pF		20		25		25	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, C_L = 50 pF		45		45		45	μs

AC Test Load

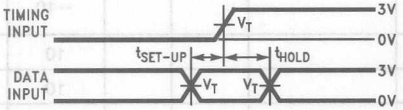


COM/IND
R1 = 200
R2 = 390

TL/L/11255-2

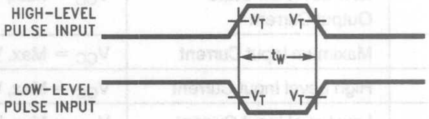
Test Waveforms

Setup and Hold



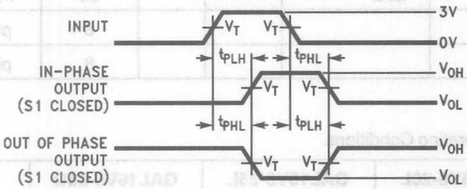
TL/L/11255-3

Pulse Width



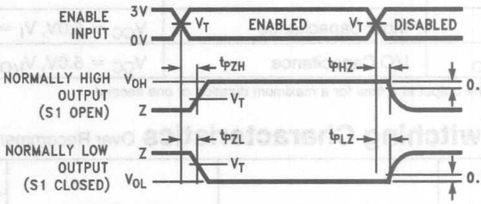
TL/L/11255-4

Propagation Delay



TL/L/11255-5

Enable and Disable



TL/L/11255-6

Notes:

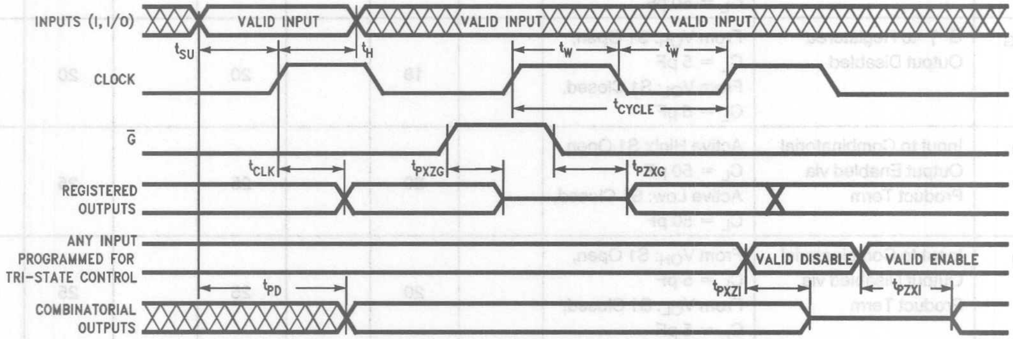
C_L includes probe and jig capacitance.

V_T = 1.5V.

Test inputs have rise and fall times of 3 ns between 0.3V and 2.7V.

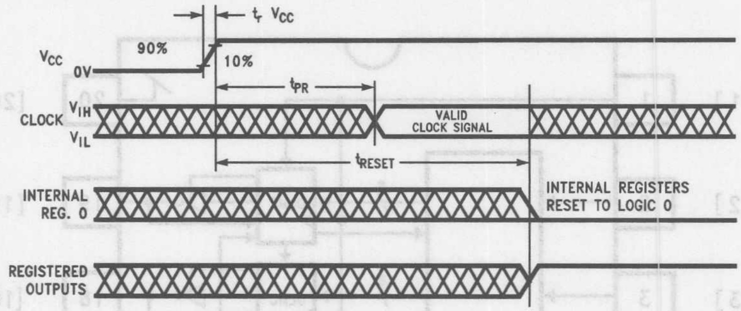
In the examples above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Switching Waveforms



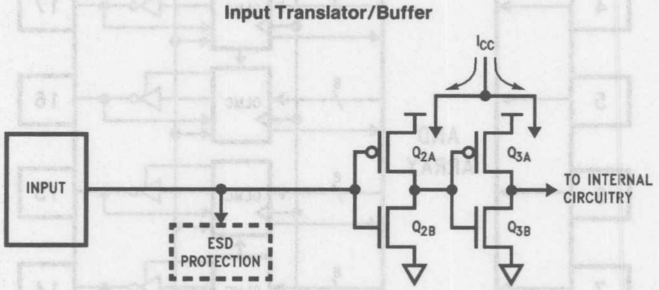
TL/L/11255-7

Power-Up Reset Waveforms



TL/L/11255-8

Input Schematic



TL/L/11255-9

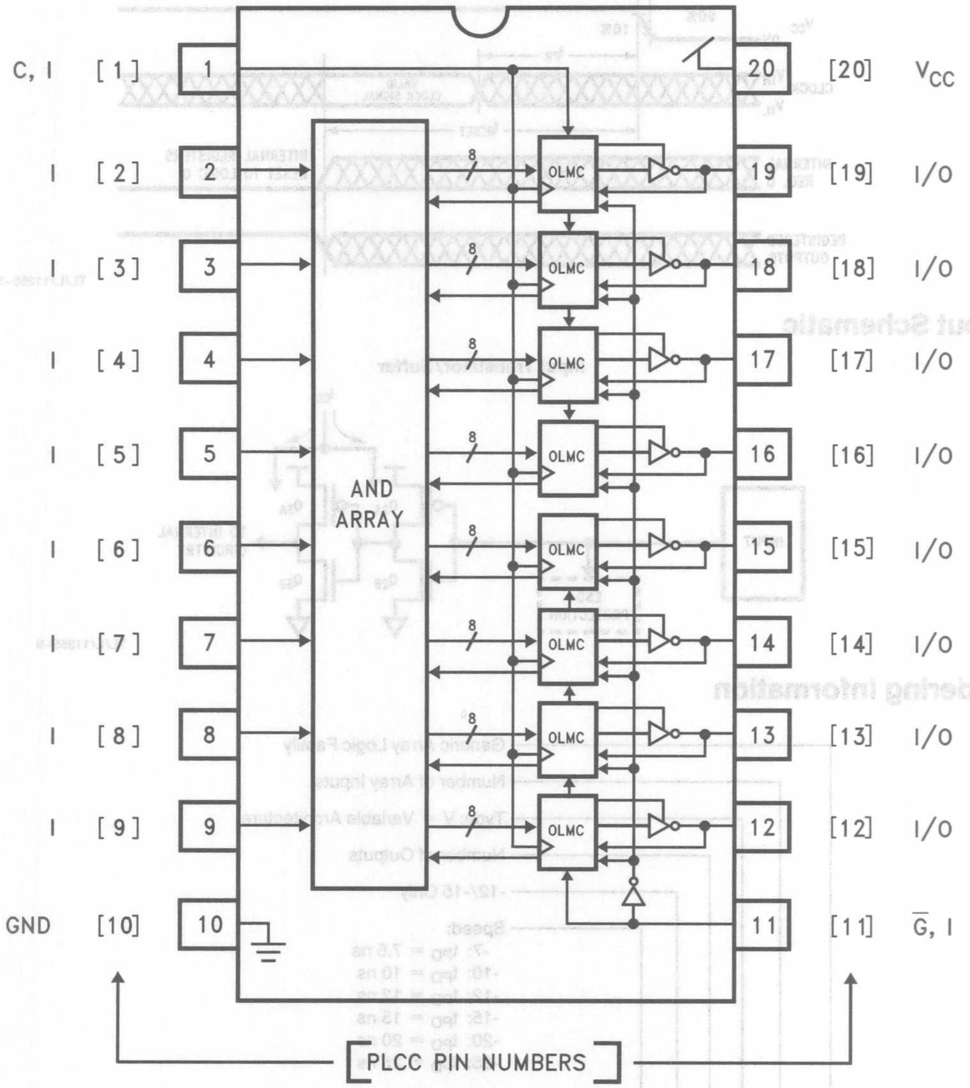
Ordering Information

GAL 16 V 8 A -7 L N C

- GAL: Generic Array Logic Family
- 16: Number of Array Inputs
- V: Type: V = Variable Architecture
- 8: Number of Outputs
- A: -12/-15 Only
- -7: Speed:
  - -7: t_{PD} = 7.5 ns
  - -10: t_{PD} = 10 ns
  - -12: t_{PD} = 12 ns
  - -15: t_{PD} = 15 ns
  - -20: t_{PD} = 20 ns
  - -25: t_{PD} = 25 ns
- L: L = Low Power
  Q = Quarter Power
- N: Package Type:
  N = 20-Pin Plastic DIP
  V = 20-Lead Plastic Chip Carrier
- C: Temperature Range:
  C = Commercial (0°C to +75°C)
  I = Industrial (−40°C to +85°C)

THE GAL16V8A-10L HAS BEEN RENAMED GAL16V8-10L. THERE WERE NO SPECIFICATION CHANGES ASSOCIATED WITH THIS NAME CHANGE.

GAL16V8 Block Diagram—DIP Connections



TL/L/11255–10

FIGURE 1

Functional Description

The GAL logic array consists of a programmable AND array with fixed OR-gate connections, similar to the bipolar PAL architecture. The logic array is organized as 16 complementary input lines crossing 64 "product term" lines with a programmable E^{2}PROM cell at each intersection (2048 cells). Each programmable cell may establish a connection between an input line (true or complement phase of an array input signal) and a product term. A product term is satisfied (logically true) while all of the input lines "connected" to it are in the high logic state.

The 64 product terms are organized into eight output groups with eight terms each. Seven or eight of the product terms in each output group feed into an OR-gate to produce each output logic function; one of the product terms may instead be used to control the associated TRI-STATE device output. The fundamental transfer function of each GAL output is the familiar Boolean sum-of-products. Design development software is available which accepts Boolean equations and converts them automatically into GAL programming patterns.

As shown in the GAL16V8 Block Diagram *(Figure 1)*, a total of eight output logic functions are available. Each of the AND/OR logic functions feeds into an "output logic macrocell" (OLMC). The eight OLMCs control the flow of input and output signals between the logic array and the device's I/O pins.

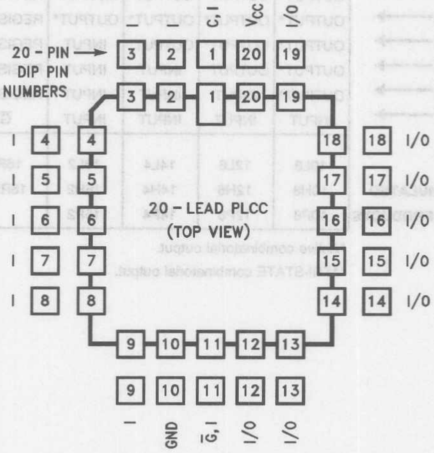
Under control of an OLMC, each output may be designated either registered or combinatorial (non-registered). In the registered output configuration, the logic function output passes through a D-type flip-flop triggered by the rising edge of the clock input. Additionally, the logic function's output polarity may be designated active-low or active-high (adjusted before the register, if present). OLMC options such as these are selected using a set of programmable architecture control cells. These architecture cells are normally configured automatically by the development software or programming hardware.

All of the possible I/O configurations of the GAL16V8 are classified into three basic modes: "Small-PAL" mode, "Registered-PAL" mode and "Medium-PAL" mode. These modes correspond to the architectures of the PAL families which the GAL16V8 can emulate. The modes determine the mixture of OLMC configurations which can be selected for the device. The OLMC Selection table (Table I) lists which functions can be selected on the device pin* 1 and pins* 11 through 19 for each of the three modes. The logic diagrams in *Figure 3* illustrate these OLMC functions.

"OUTPUT" represents the always-active combinatorial output configuration available in the "Small-PAL" mode. "REGISTER" is the registered output with register feedback available in the "Registered-PAL" mode. "I/O" is the combinatorial bidirectional I/O available in "Registered-PAL" and "Medium-PAL" modes. "TRI-STATE" is the TRI-STATE combinatorial output function appearing on pins* 12 and 19 in the "Medium-PAL" mode. "INPUT" in Table I denotes an OLMC used as a dedicated input only.

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8.

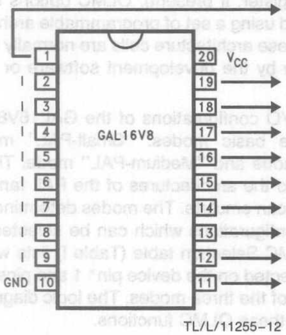
20-Lead PLCC Connection Diagram



TL/L/11255-11

FIGURE 2

OLMC Selection Table



TL/L/11255-12

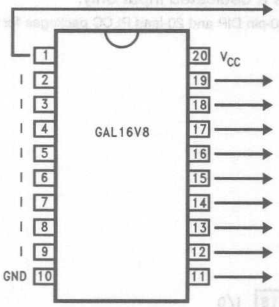
TABLE I

"Small-PAL" Mode	"Registered-PAL" Mode	"Medium-PAL" Mode
INPUT	CLOCK	INPUT
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT	OUPUT ENABLE ($\overline{G}$)	INPUT

*Active combinatorial output

**TRI-STATE combinatorial output

PAL Replacement Configurations



TL/L/11255-13

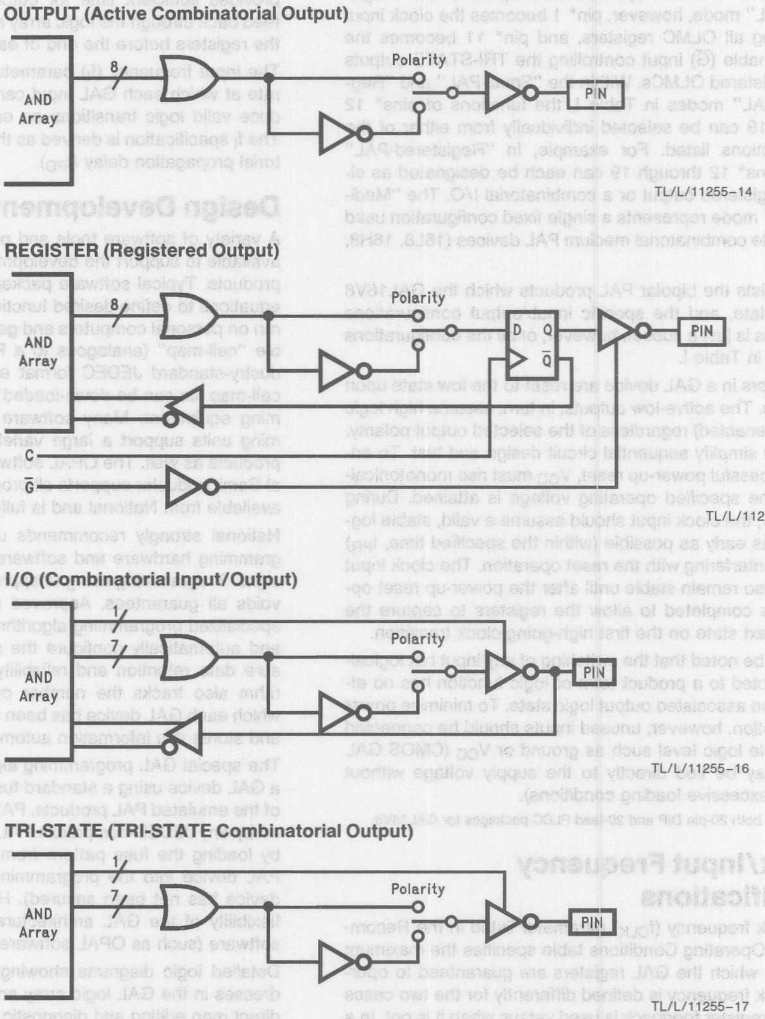
TABLE II

	"Small PAL" Mode				"Registered-PAL" Mode			"Medium-PAL" Mode
	INPUT	INPUT	INPUT	INPUT	CLOCK	CLOCK	CLOCK	INPUT
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	INPUT	INPUT	INPUT	INPUT	$\overline{G}$	$\overline{G}$	$\overline{G}$	INPUT
EMULATED PAL PRODUCTS	10L8 10H8 10P8	12L6 12H6 12P6	14L4 14H4 14P4	16L2 16H2 16P2	16R8 16RP8	16R6 16RP6	16R4 16RP4	16L8 16H8 16P8

*Active combinatorial output.

**TRI-STATE combinatorial output.

OLMC Configurations


OUTPUT (Active Combinatorial Output)

TL/L/11255–14

REGISTER (Registered Output)

TL/L/11255–15

I/O (Combinatorial Input/Output)

TL/L/11255–16

TRI-STATE (TRI-STATE Combinatorial Output)

TL/L/11255–17

FIGURE 3

Functional Description (Continued)

In the "Small-PAL" and "Medium-PAL" modes (Table I), pins* 1 and 11 are always dedicated inputs. In the "Registered-PAL" mode, however, pin* 1 becomes the clock input controlling all OLMC registers, and pin* 11 becomes the output enable ($\overline{G}$) input controlling the TRI-STATE outputs of all registered OLMCs. Within the "Small-PAL" and "Registered-PAL" modes in Table I, the functions of pins* 12 through 19 can be selected individually from either of the two functions listed. For example, in "Registered-PAL" mode, pins* 12 through 19 can each be designated as either a registered output or a combinatorial I/O. The "Medium-PAL" mode represents a single fixed configuration used to emulate combinatorial medium PAL devices (16L8, 16H8, 16P8).

Table II lists the bipolar PAL products which the GAL16V8 can emulate, and the specific input/output configurations used. This is just a subset, however, of all the configurations provided in Table I.

All registers in a GAL device are reset to the low state upon power-up. The active-low outputs, in turn, assume high logic levels (if enabled) regardless of the selected output polarity. This may simplify sequential circuit design and test. To ensure successful power-up reset, V_{CC} must rise monotonically until the specified operating voltage is attained. During power-up, the clock input should assume a valid, stable logic state as early as possible (within the specified time, t_{PR}) to avoid interfering with the reset operation. The clock input should also remain stable until after the power-up reset operation is completed to allow the registers to capture the proper next state on the first high-going clock transition.

It should be noted that the switching of any input not logically connected to a product term or logic function has no effect on the associated output logic state. To minimize power consumption, however, unused inputs should be connected to a stable logic level such as ground or V_{CC} (CMOS GAL inputs may be tied directly to the supply voltage without causing excessive loading conditions).

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8.

Clock/Input Frequency Specifications

The clock frequency (f_{CLK}) parameter listed in the Recommended Operating Conditions table specifies the maximum speed at which the GAL registers are guaranteed to operate. Clock frequency is defined differently for the two cases in which register feedback is used versus when it is not. In a data-path type application, when the logic functions fed into the registers are not dependent on register feedback from the previous cycle (i.e. based only on external inputs), the minimum required cycle period (f_{CLK}^{-1} without feedback) is defined as the greater of the minimum clock period (t_w high + t_w low) and the minimum "data window" period (t_{SU} + t_H). This assumes optimal alignment between data inputs and the clock input. In sequential logic applications such as state machines, the minimum required cycle period (t_{CYCLE} = f_{CLK}^{-1} with feedback) is defined as t_{CLK} + t_{SU}. This provides sufficient time for outputs from the registers to feed back through the logic array and set up on the inputs to the registers before the end of each cycle.

The input frequency (f_I) parameter specifies the maximum rate at which each GAL input can be toggled and still produce valid logic transitions on each combinatorial output. The f_I specification is derived as the inverse of the combinatorial propagation delay (t_{PD}).

Design Development Support

A variety of software tools and programming equipment is available to support the development of designs using GAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate a JEDEC-compatible "cell-map" (analogous to a PAL "fuse-map"). The industry-standard JEDEC format ensures that the resulting cell-map file can be down-loaded into a variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible.

National strongly recommends using only approved programming hardware and software for developing GAL designs. Programming using unapproved equipment generally voids all guarantees. Approved programmers incorporate specialized programming algorithms that program the array and automatically configure the architecture cells. To ensure data retention and reliability, the programming algorithm also tracks the number of programming cycles to which each GAL device has been subjected since shipment, and stores this information automatically in the device.

The special GAL programming algorithm can also program a GAL device using a standard fuse-map developed for any of the emulated PAL products. PAL fuse-maps can be created by any JEDEC-compatible PAL development software or by loading the fuse pattern from an existing programmed PAL device into the programming unit (provided the PAL device has not been secured). However, to utilize the full flexibility of the GAL architecture, true GAL development software (such as OPAL software) is recommended.

Detailed logic diagrams showing all JEDEC cell-map addresses in the GAL logic array and OLMC are provided for direct map editing and diagnostic purposes (see "Programming Details"). For a list of current software and programming support tools available for these devices, please contact your local National sales representative or distributor. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

Security Cell

A security cell is provided on all GAL16V8 devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the array. The security cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed.

Electronic Signature

Each GAL device contains an electronic signature word consisting of 64 bits of reprogrammable memory. The electronic signature word can be programmed to contain any identification information desired by the user. Some uses include pattern identification labels, revision numbers, dates, inventory control information, etc. The data stored in the electronic signature word has no effect on the functionality of the device. The information is read out of the device using the normal program verification procedure provided by the programming equipment. The information may be accessed at any time independent of the state of the security cell. National's OPAL development software allows electronic signature data to be entered by the user and downloaded to the programming equipment.

Bulk Erase

The programming equipment automatically performs a bulk erase operation prior to each programming operation. No special erase operation need be performed by the user. Bulk erase clears the logic array, architecture cells, security cell, and electronic signature information. The GAL device is thereby reverted back to its virgin state.

Latch-Up Protection

GAL devices are designed with an on-chip charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

To insure that no undesired bias conditions occur with P+ diffusions, a Latch-Lock™ power-up circuitry has been developed. The drain of all P channel devices normally connected to the device supply are now connected to an alternate supply that powers up after the device N-wells have been biased and the substrate has reached its negative clamp value. This prevents any hazardous bias conditions from developing in the power-up sequence. After power-up is complete, the Latch-Lock circuitry becomes dormant until a full power-down has occurred; this eliminates the chance of an unwanted P channel power-down during device operation.

Manufacturer Testing

Because of EECMOS technology, GAL devices can be reprogrammed in milliseconds. This allows each device to be completely tested by the manufacturer using numerous logic array and architecture patterns prior to shipping. Every programmable cell and every logic path through every device is fully tested for programmability, functionality and performance to all AC and DC parameters. The customer can therefore expect 100% programming and functional yield and 100% compliance of all GAL products to datasheet specifications.

The testing procedure performed on all GAL devices by the manufacturer tests all aspects of device operation. Extensive testing of all programmable cells in the device include margin testing, internal verify, and program retention during high-temperature bake. All DC and AC parameters are tested at hot and cold temperatures using a variety of worst case logic and signal patterns. Functional tests include reprogramming each OLMC to all valid architectural configurations.

Register Preload

The register preload feature allows OLMC registers to be directly loaded with any desired data pattern. It also allows the present state of OLMC registers to be examined regardless of TRI-STATE control conditions. This simplifies testing of devices after programming. A device may be put into any desired register state at any point during the functional test sequence. The test sequence may then be resumed to verify proper next-state transitions. This allows complete verification of sequential logic circuits, including states that are normally impossible or difficult to reach. It may also shorten the overall test time significantly.

Register preload is not an operational mode and is not intended for board-level testing because elevated voltage levels must be applied to the device. The programming equipment normally provides the register preload capability as part of its functional test facility. Note that the testing of GAL devices after programming by the user may be considered unnecessary because all EECMOS GAL products are completely tested by the manufacturer, guaranteeing 100% post-programming functional yield.

The register preload algorithm is described for those users who wish to test programmed GAL devices using test equipment other than approved GAL programming equipment. As shown in the Register Preload Waveform in *Figure 5*, the preload sequence must not begin until the normal power-up reset operation has completed (after time t_{RESET}). The device is placed into preload mode by raising the "PRLD" input (pin* 11) to voltage V_{IES}, as specified in the Register Preload Specifications (Table III).

*Applies to both 20-pin DIP and 20-lead PLCC Packages for GAL16V8.

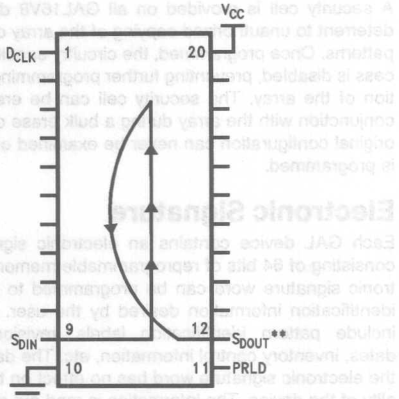
Register Preload (Continued)

To preload the OLMC registers, a series of data bits are shifted into the device on the "S_{DIN}" input (pin* 9), one bit for each OLMC in which registered output has been selected. (Non-registered OLMCs are bypassed.) The shift sequence is clocked by the rising edge of the "D_{CLK}" input (pin* 1). The data stream is shifted in through the registered OLMC with the lowest corresponding pin number, and then "upward" through all remaining registered OLMCs in pin-number ascending order. Therefore, the first data bit in the series is ultimately loaded into the registered OLMC with the highest corresponding pin number, as shown in *Figure 4*.

As the data series is shifted into the S_{DIN} input, the contents of all registers (in registered OLMCs) are shifted "upward" and out onto the "S_{DOUT}" output (pin* 12). Complete present-state information can be examined in this manner. Test fixtures can be devised to test several GAL devices in which the S_{DOUT} pin of each chip is connected to the S_{DIN} pin of the next, and all preload and present-state data can be shifted around a single serial loop.

Note that when shifting register data into S_{DIN} or out of S_{DOUT}, V_{IL}/V_{OL} = register reset (0), and V_{IH}/V_{OH} = register set (1). These 0 and 1 register states are always inverted (active-low) on the normal output pins regardless of the selected output polarity (polarity affects logic function values before register inputs).

*Applies to both 20-pin DIP and 20-lead PLCC Packages for GAL16V8.



TL/L/11255-18

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

FIGURE 4. Output Register Preload Pinout

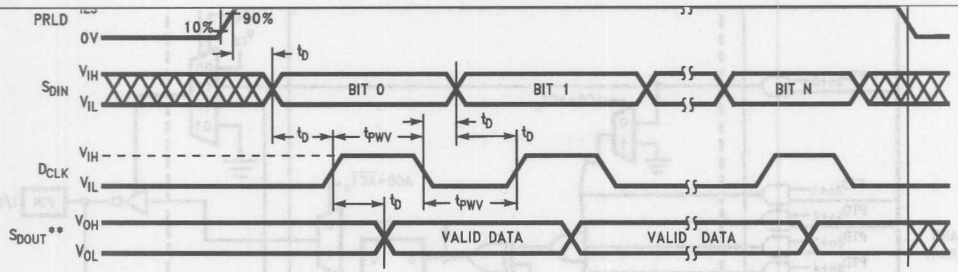
Register Preload Specifications

TABLE III

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input Voltage (High)		2.40		V_{CC}	V
V_{IL}	Input Voltage (Low)		0.00		0.50	V
V_{IES}	Registered Preload Input Voltage		11.5	12	12.5	V
V_{OH}	Output Voltage (High) (Note 1)				V_{CC}	V
V_{OL}	Output Voltage (Low) (Note 1)	$I_{OL} \leq 12$ mA	0.00		0.50	V
I_{IH}, I_{IL}	Input Current (Programming)			±1	±10	μA
I_{OH}	High Level Output Current (Note 1)	$V_{OH} \leq V_{CC}$			10	μA
t_{PWV}	Verify Pulse Width		1	5	10	μs
t_D	Pulse Sequence Delay		1	5	10	μs
t_{RESET}	Register Reset Time from Valid V_{CC}				45	μs

Note 1: The S_{DOUT} output buffer is an open drain output. This pin should be terminated to V_{CC} with a 10k resistor.

FIGURE 5



TL/L/11255–19

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

Programming Details

Understanding the information in this section is not essential when using approved programming equipment and software for developing GAL designs. This is a more thorough disclosure of the GAL architecture provided for direct JEDEC cell-map editing and diagnostic purposes. This section alone, however, does not contain sufficient information to implement the GAL programming algorithm. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

As mentioned in the Functional Description, the OLMC is responsible for selecting input and/or output paths, registered vs. combinatorial outputs, active-high or low polarity, and common vs. locally-controlled TRI-STATE control. Additionally, the OLMCs select between alternate logic array input paths to maintain JEDEC cell-map compatibility with either "small-PAL" or "medium-PAL" logic arrays.

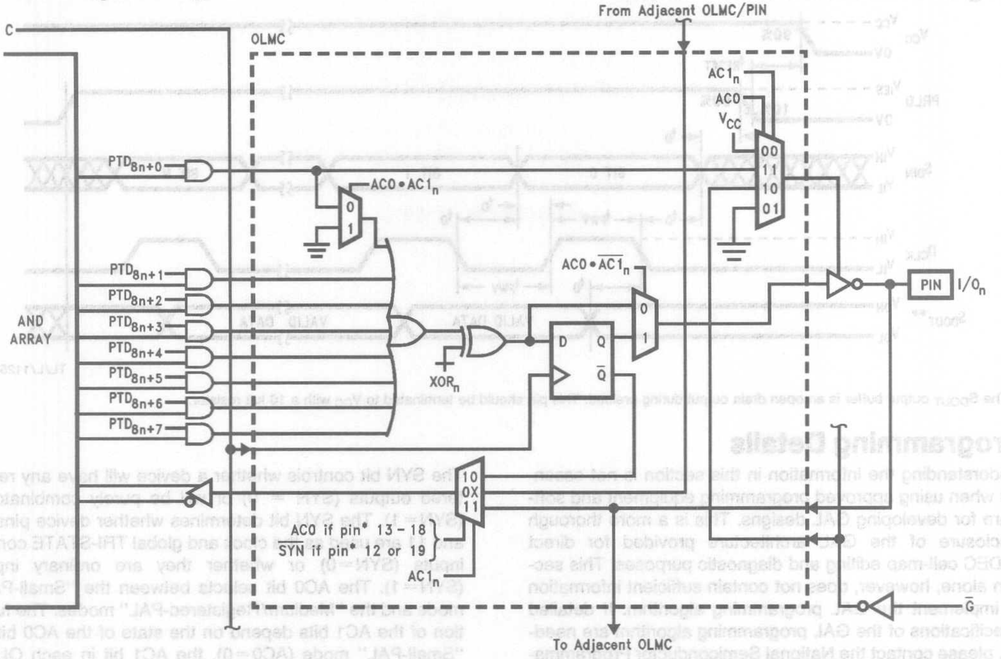
The various configurations of the OLMCs are controlled by a set of programmable "architecture" cells, separate from the logic-defining array cells. Each GAL device contains two "global" architecture cells, "SYN" and "AC0", which affect all OLMCs. Each of the device's eight OLMCs also contains two "local" cells, "AC1" and "XOR". The OLMC Logic Diagram in *Figure 6* shows how the architecture cells select the different paths through the OLMC.

The SYN bit controls whether a device will have any registered outputs (SYN = 0) or will be purely combinatorial (SYN = 1). The SYN bit determines whether device pins* 1 and 11 are used as the clock and global TRI-STATE control inputs (SYN = 0) or whether they are ordinary inputs (SYN = 1). The AC0 bit selects between the "Small-PAL" mode and the "Medium/Registered-PAL" modes. The function of the AC1 bits depend on the state of the AC0 bit. In "Small-PAL" mode (AC0 = 0), the AC1 bit in each OLMC determines whether the associated device pin is an output (AC1 = 0) or an input (AC1 = 1). In "Registered-PAL" mode (AC0 = 1), the AC1 bit determines whether each OLMC is registered (AC1 = 0) or combinatorial (AC1 = 1). In "Medium-PAL" mode (AC0 = 1), the AC1 bits in all OLMCs must be set to 1 (combinatorial). All of the valid architecture bit configurations are shown in the OLMC Architecture table (Table IV), which has the same familiar format used in the OLMC Selection table (Table I).

Independent of SYN, AC0 and the AC1 bits, the XOR bit in each OLMC selects between active-low (XOR = 0) or active-high (XOR = 1) output polarity.

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8.

OLMC Logic Diagram



TL/L/11255-20

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8.

FIGURE 6

OLMC Architecture Programming

TABLE IV

	"Small-PAL" Mode			"Registered-PAL" Mode			"Medium-PAL" Mode	
	Function		JEDEC Input Line #s (Note 1)	Function		JEDEC Input Line #s (Note 1)	Function	JEDEC Input Lines #s (Note 1)
Pin 1	INPUT	INPUT	2,3	CLOCK	CLOCK		INPUT	2,3
*** Pin 19	I/O	INPUT	6,7	REGISTER	I/O	2,3	TRI-STATE**	
*** Pin 18	I/O	INPUT	10,11	REGISTER	I/O	6,7	I/O	6,7
*** Pin 17	I/O	INPUT	14,15	REGISTER	I/O	10,11	I/O	10,11
*** Pin 16	OUTPUT*	NC		REGISTER	I/O	14,15	I/O	14,15
*** Pin 15	OUTPUT*	NC		REGISTER	I/O	18,19	I/O	18,19
*** Pin 14	I/O	INPUT	18,19	REGISTER	I/O	22,23	I/O	22,23
*** Pin 13	I/O	INPUT	22,23	REGISTER	I/O	26,27	I/O	26,27
*** Pin 12	I/O	INPUT	26,27	REGISTER	I/O	30,13	TRI-STATE**	
Pin 11	INPUT	INPUT	30,31	$\overline{G}$	$\overline{G}$		INPUT	30,31
	$AC1_n = 0$	$AC1_n = 1$		$AC1_n = 0$	$AC1_n = 1$		$AC1_n = 1$	
	SYN = 1, AC0 = 0			SYN = 0, AC0 = 1			SYN = 1, AC0 = 1	
	All outputs are combinatorial and always active.			At least one output is registered.			All I/O pins are combinatorial.	

Note: Pin numbers above apply to both 20-pin DIP and 20-lead PLCC packages for GAL16V8.

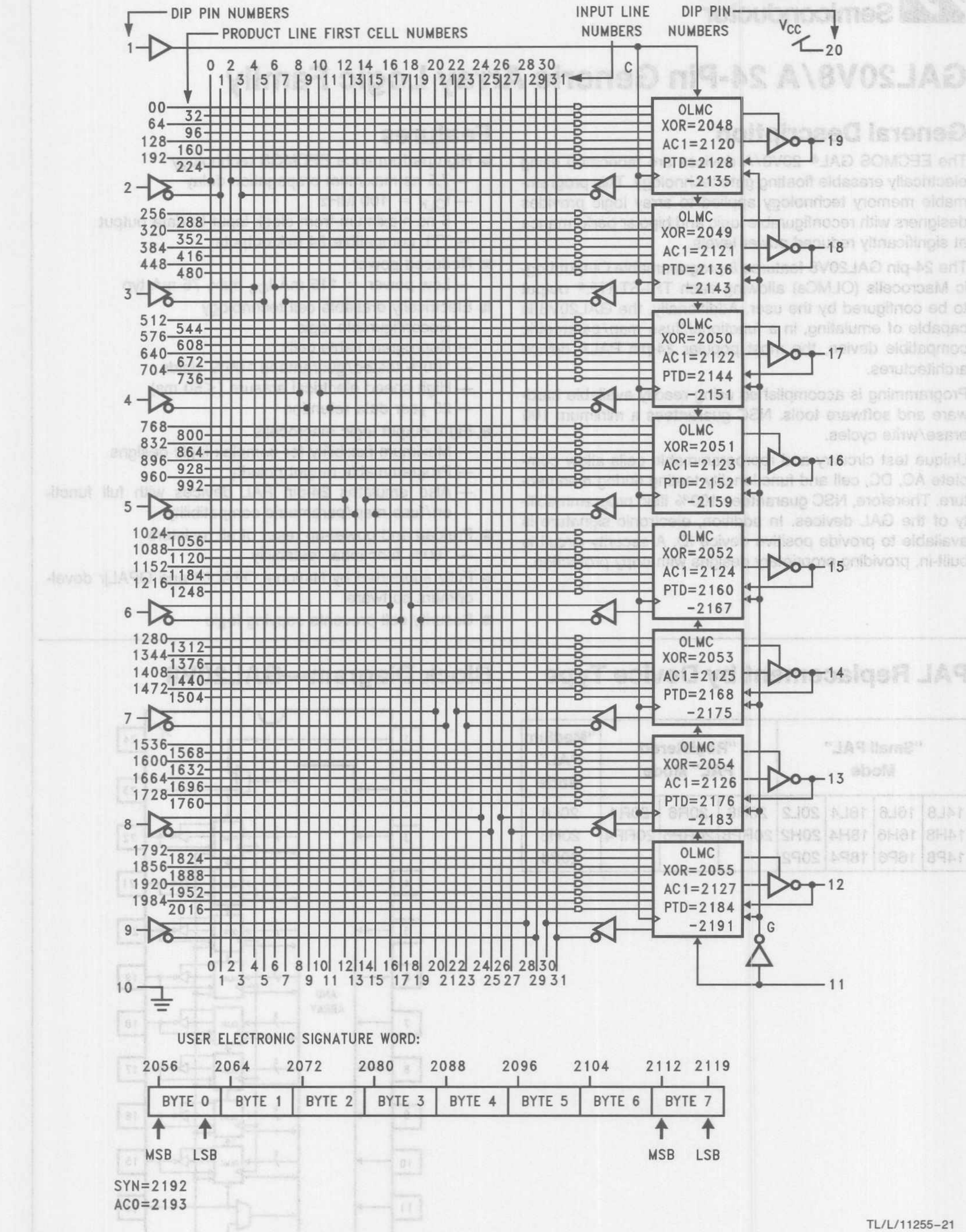
Note 1: All even and odd numbered JEDEC input line numbers correspond to true and complement array inputs, respectively.

*Active combinatorial output.

**TRI-STATE combinatorial output.

*** $AC1_n$ applies to these I/O pins only.

GAL16V8 Logic Diagram



TL/L/11255-21

JEDEC Logic Array Cell Number = Product Line First Cell Number + Input Line Number

FIGURE 7

National Semiconductor

GAL20V8/A 24-Pin Generic Array Logic Family

General Description

The EECMOS GAL® 20V8/A devices are fabricated using electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 24-pin GAL20V8 features 8 programmable Output Logic Macrocells (OLMCs) allowing each TRI-STATE® output to be configured by the user. Additionally, the GAL20V8 is capable of emulating, in a functional/fuse map/parametric compatible device, the most popular 24-pin PAL® device architectures.

Programming is accomplished using readily available hardware and software tools. NSC guarantees a minimum 100 erase/write cycles.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, NSC guarantees 100% field programmability of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

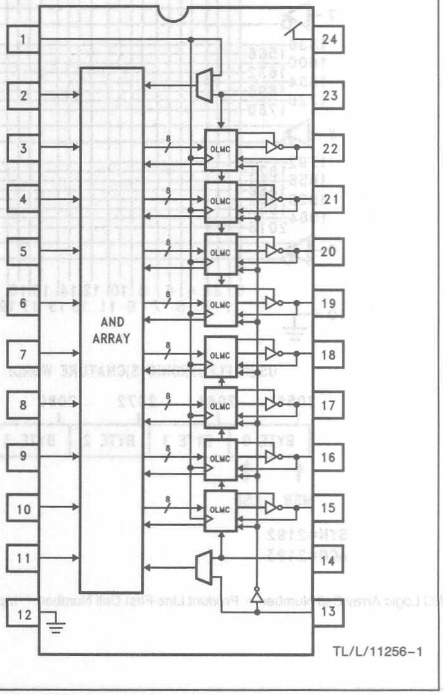
Features

- High performance EECMOS technology
  - — 7.5 ns maximum propagation delay
  - — f_{CLK} = 100 MHz
  - — 5 ns maximum from clock input to data output
  - — TTL compatible 24 mA outputs
- Reduced power
  - — Low power = 115 mA I_{CC} max, 75 mA typ
- Electrically erasable cell technology
  - — Reconfigurable logic
  - — Reprogrammable cells
  - — 100% tested/guaranteed 100% yields
  - — High speed electrical erasure (<50 ms)
  - — 20 year data retention
- Eight output logic macrocells
  - — Maximum flexibility for complex logic designs
  - — Programmable output polarity
  - — Also emulates 24-pin PAL devices with full function/fuse map/parametric compatibility
- Preload and power-up reset of all registers
  - — 100% functional testability
- Fully supported by National OPAL™ and OPALjr development software
- Security cell prevents copying logic

PAL Replacement by Device Type

"Small PAL" Mode				"Registered PAL" Mode			"Medium PAL" Mode
14L8	16L6	18L4	20L2	20R8	20R6	20R4	20L8
14H8	16H6	18H4	20H2	20RP8	20RP6	20RP4	20H8
14P8	16P6	18P4	20P2				20P8

Block Diagram—GAL20V8



GAL20V8-7/-10

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	>2000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Min	Typ	Max	Units
V_{CC}	Supply Voltage	4.75	5	5.25	V
T_A	Operating Free-Air Temperature	0	25	75	°C
T_C	Operating Case Temperature				°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL20V8-7L		GAL20V8-10L		Units
			Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		7		10		ns
t_H	Hold Time (Input after Clock)		0		0		ns
t_w	Clock Pulse Width (High/Low)		5		8		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		12		17		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		83.3		58.8	MHz
		Without Feedback		100		62.5	
f_I	Input Frequency (Note 5)			133.3		100	
t_{PR}	Clock Valid after Power-Up			100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) = $(t_{CYCLE})^{-1}$
f_{CLK} (without feedback) = $(2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL20V8-7/-10

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Min	Typ	Max	Units
V_{IH}	High Level Input Voltage			2.0		$V_{CC} + 1$	V
V_{IL}	Low Level Input Voltage			−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	$I_{OH} = -3.2$ mA	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 24 mA			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)				10	µA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND				−10	µA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC} (Max)				10	µA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC} (Max)				10	µA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND				−10	µA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND		−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max			75	115	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V			5	8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V			5	8	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL20V8-7		GAL20V8-10		Units
			Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, C_L = 50 pF		7.5		10	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, C_L = 50 pF		5		7	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		6		10	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}; S1 Open, C_L = 5 pF From V_{OL}; S1 Closed, C_L = 5 pF		6		10	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		9		10	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}; S1 Open, C_L = 5 pF From V_{OL}; S1 Closed, C_L = 5 pF		9		10	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, C_L = 50 pF		45		45	µs

GAL20V8A-12/-15

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance C_{ZAP} = 100 pF R_{ZAP} = 1500Ω Test Method: Human Body Model Test Specification: NSC SOP-5-028 Rev. C	1000V

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL20V8A-12L COM		GAL20V8A-15L COM/IND		Units
			Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		10		12		ns
t_H	Hold Time (Input after Clock)		0		0		ns
t_w	Clock Pulse Width (High/Low)		8		8		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		20		22		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		50		45.5	
		Without Feedback		62.5		62.5	MHz
f_I	Input Frequency (Note 5)			83.3		66.6	
t_{PR}	Clock Valid after Power-Up			100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) = $(t_{CYCLE})^{-1}$
f_{CLK} (without feedback) = $(2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL20V8A-12/-15

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage			2.0		$V_{CC} + 1$	V
V_{IL}	Low Level Input Voltage			−0.5		0.8	V
V_{OH}	High Level Output Voltage	$V_{CC} = \text{Min}$ $I_{OH} = -3.2$ mA		2.4			V
V_{OL}	Low Level Output Voltage	$V_{CC} = \text{Min}$ $I_{OL} = 24$ mA				0.5	V
I_{OZH}	High Level Off State Output Current	$V_{CC} = \text{Max}, V_O = V_{CC}$ (Max)				10	μA
I_{OZL}	Low Level Off State Output Current	$V_{CC} = \text{Max}, V_O = \text{GND}$				−10	μA
I_I	Maximum Input Current	$V_{CC} = \text{Max}, V_I = V_{CC}$ (Max)				10	μA
I_{IH}	High Level Input Current	$V_{CC} = \text{Max}, V_I = V_{CC}$ (Max)				10	μA
I_{IL}	Low Level Input Current	$V_{CC} = \text{Max}, V_I = \text{GND}$				−10	μA
I_{OS}*	Output Short Circuit Current	$V_{CC} = 5.0\text{V}, V_O = \text{GND}$		−30		−150	mA
I_{CC}	Supply Current	$f = 25$ MHz, $V_{CC} = \text{Max}$	COM			90	mA
			IND			130	mA
C_I	Input Capacitance	$V_{CC} = 5.0\text{V}, V_I = 2.0\text{V}$				8	pF
$C_{I/O}$	I/O Capacitance	$V_{CC} = 5.0\text{V}, V_{I/O} = 2.0\text{V}$				10	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL20V8A-12L		GAL20V8A-15L		Units
			COM		COM/IND		
			Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, $C_L = 50$ pF		12		15	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, $C_L = 50$ pF		10		10	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High; S1 Open, $C_L = 50$ pF Active Low; S1 Closed, $C_L = 50$ pF		10		15	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}; S1 Open, $C_L = 5$ pF From V_{OL}; S1 Closed, $C_L = 5$ pF		10		15	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High; S1 Open, $C_L = 50$ pF Active Low; S1 Closed, $C_L = 50$ pF		12		15	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}; S1 Open, $C_L = 5$ pF From V_{OL}; S1 Closed, $C_L = 5$ pF		12		15	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, $C_L = 50$ pF		45		45	μs

GAL20V8-20/-25

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C

Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance C_{ZAP} = 100 pF R_{ZAP} = 1500Ω Test Method: Human Body Model Test Specification: NSC SOP-5-028 Rev. C	1000V

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL20V8-20L		GAL20V8-25L		GAL20V8-25Q		Units
			COM		COM		COM		
			Min	Max	Min	Max	Min	Min	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		15		15		15		ns
t_H	Hold Time (Input after Clock)		0		0		0		ns
t_w	Clock Pulse Width (High/Low)		12		12		12		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		27		27		27		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		37		37		37	MHz
		Without Feedback		41.66		41.66		41.66	
f_I	Input Frequency (Note 5)			50		50		50	
t_{PR}	Clock Valid after Power-Up			100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) $= (t_{CYCLE})^{-1}$
f_{CLK} (without feedback) $= (2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL20V8-20/-25

Electrical Characteristics Over Recommended Operating Conditions

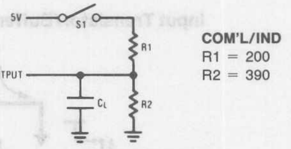
Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		$V_{CC} + 1$	V
V_{IL}	Low Level Input Voltage				−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA		2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 24 mA				0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	-20L, -25L				90	mA
			-25Q				55	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					10	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL20V8-20L		GAL20V8-25L		GAL20V8-25Q		Units
			COM		COM		COM		
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, C_L = 50 pF		20		25		25	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, C_L = 50 pF		12		12		12	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		18		20		20	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}; S1 Open, C_L = 5 pF From V_{OL}; S1 Closed, C_L = 5 pF		18		20		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		20		25		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}; S1 Open, C_L = 5 pF From V_{OL}; S1 Closed, C_L = 5 pF		20		25		25	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, C_L = 50 pF		45		45		45	μs

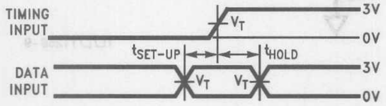
AC Test Load



TL/L/11256-2

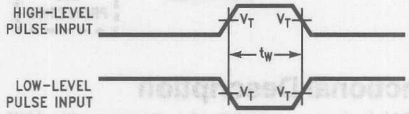
Test Waveforms

Setup and Hold



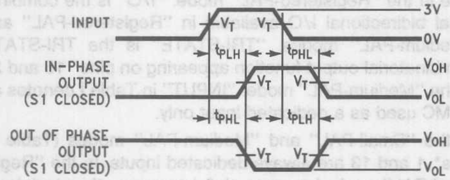
TL/L/11256-3

Pulse Width



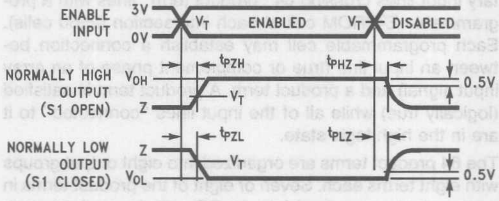
TL/L/11256-4

Propagation Delay



TL/L/11256-5

Enable and Disable



TL/L/11256-6

Notes:

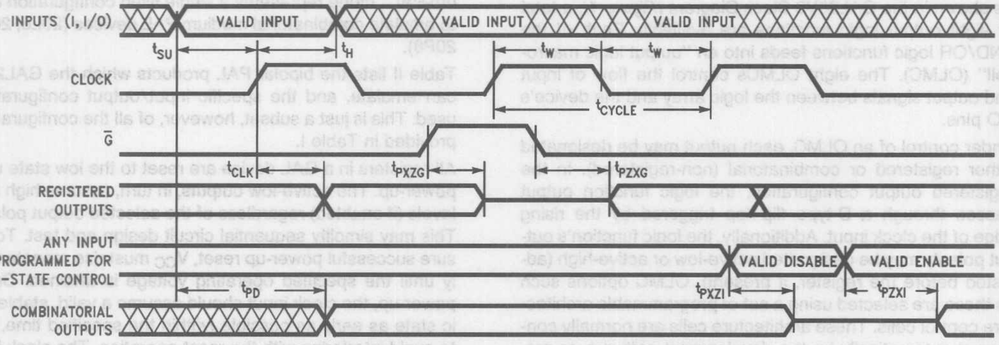
C_L includes probe and jig capacitance.

$V_T = 1.5V$.

Test inputs have rise and fall times of 3 ns between 0.3V and 2.7V.

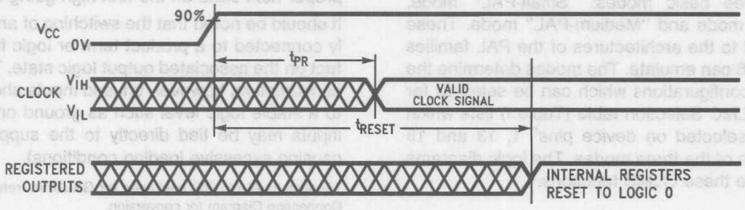
In the example above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Switching Waveforms



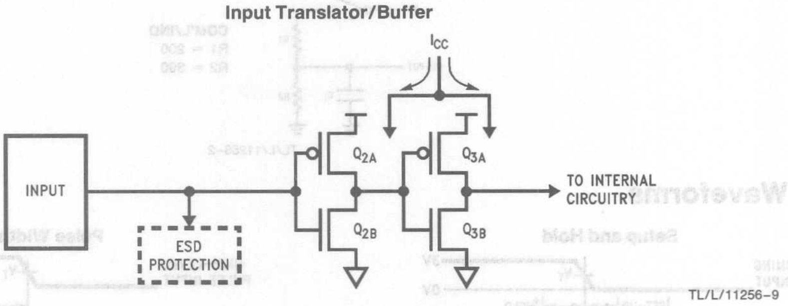
TL/L/11256-7

Power-Up Reset Waveforms



TL/L/11256-8

Input Schematic



Functional Description

The GAL logic array consists of a programmable AND array with fixed OR-gate connections, similar to the bipolar PAL architecture. The logic array is organized as 20 complementary input lines crossing 64 "product term" lines with a programmable E^{2}PROM cell at each intersection (2560 cells). Each programmable cell may establish a connection between an input line (true or complement phase of an array input signal) and a product term. A product term is satisfied (logically true) while all of the input lines "connected" to it are in the high logic state.

The 64 product terms are organized into eight output groups with eight terms each. Seven or eight of the product terms in each output group feed into an OR-gate to produce each output logic function; one of the product terms may instead be used to control the associated TRI-STATE device output. The fundamental transfer function of each GAL output is the familiar Boolean sum-of-products. Design development software is available which accepts Boolean equations and converts them automatically into GAL programming patterns.

As shown in the GAL20V8 Block Diagram *(Figure 1)*, a total of eight output logic functions are available. Each of the AND/OR logic functions feeds into an "output logic macrocell" (OLMC). The eight OLMCs control the flow of input and output signals between the logic array and the device's I/O pins.

Under control of an OLMC, each output may be designated either registered or combinatorial (non-registered). In the registered output configuration, the logic function output passes through a D-type flip-flop triggered by the rising edge of the clock input. Additionally, the logic function's output polarity may be designated active-low or active-high (adjusted before the register, if present). OLMC options such as these are selected using a set of programmable architecture control cells. These architecture cells are normally configured automatically by the development software or programming hardware.

All of the possible I/O configurations of the GAL20V8 are classified into three basic modes: "Small-PAL" mode, "Registered-PAL" mode and "Medium-PAL" mode. These modes correspond to the architectures of the PAL families which the GAL20V8 can emulate. The modes determine the mixture of OLMC configurations which can be selected for the device. The OLMC Selection table (Table I) lists which functions can be selected on device pins* 1, 13 and 15 through 22 for each of the three modes. The logic diagrams in *Figure 3* illustrate these OLMC functions.

"OUTPUT" represents the always-active combinatorial output configuration available in the "Small-PAL" mode. "REGISTER" is the registered output with register feedback available in the "Registered-PAL" mode. "I/O" is the combinatorial bidirectional I/O available in "Registered-PAL" and "Medium-PAL" modes. "TRI-STATE" is the TRI-STATE combinatorial output function appearing on pins* 15 and 22 in the "Medium-PAL" mode. "INPUT" in Table I denotes an OLMC used as a dedicated input only.

In the "Small-PAL" and "Medium-PAL" modes (Table I), pins* 1 and 13 are always dedicated inputs. In the "Registered-PAL" mode, however, pin* 1 becomes the clock input controlling all OLMC registers, and pin* 13 becomes the output enable ($\overline{G}$) input controlling the TRI-STATE outputs of all registered OLMCs. Within the "Small-PAL" and "Registered-PAL" modes in Table I, the functions of pins* 15 through 22 can be selected individually from either of the two functions listed. For example, in "Registered-PAL" mode, pins* 15 through 22 can each be designated as either a registered output or a combinatorial I/O. The "Medium-PAL" mode represents a single fixed configuration used to emulate combinatorial medium PAL devices (20L8, 20H8, 20P8).

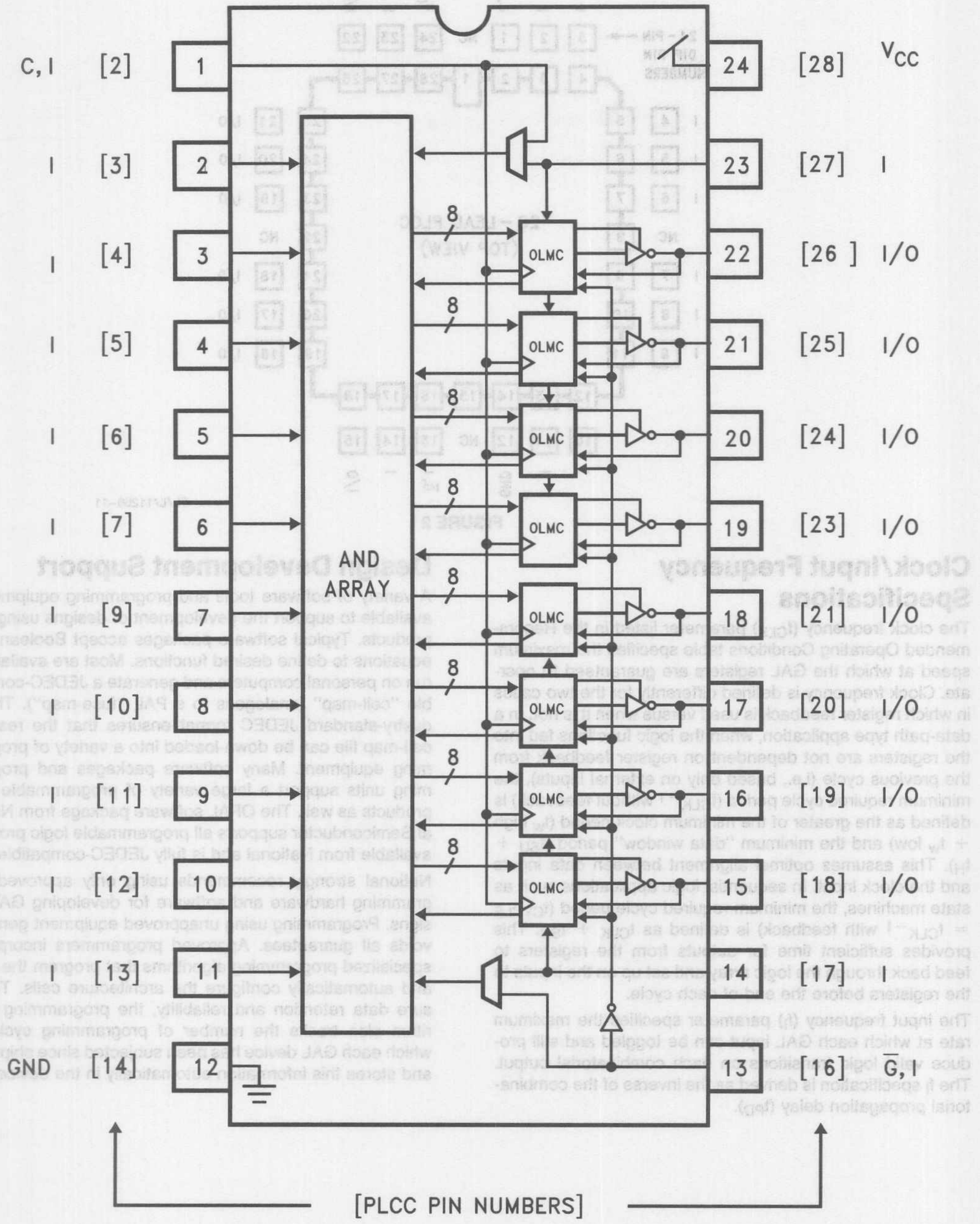
Table II lists the bipolar PAL products which the GAL20V8 can emulate, and the specific input/output configurations used. This is just a subset, however, of all the configurations provided in Table I.

All registers in a GAL device are reset to the low state upon power-up. The active-low outputs, in turn, assume high logic levels (if enabled) regardless of the selected output polarity. This may simplify sequential circuit design and test. To ensure successful power-up reset, V_{CC} must rise monotonically until the specified operating voltage is attained. During power-up, the clock input should assume a valid, stable logic state as early as possible (within the specified time, t_{PR}) to avoid interfering with the reset operation. The clock input should also remain stable until after the power-up reset operation is completed to allow the registers to capture the proper next state on the first high-going clock transition.

It should be noted that the switching of any input not logically connected to a product term or logic function has no effect on the associated output logic state. To minimize power consumption, however, unused inputs should be connected to a stable logic level such as ground or V_{CC} (CMOS GAL inputs may be tied directly to the supply voltage without causing excessive loading conditions).

*Applies to 24-pin DIP packages for GAL20V8; refer to the 28-lead PLCC Connection Diagram for conversion.

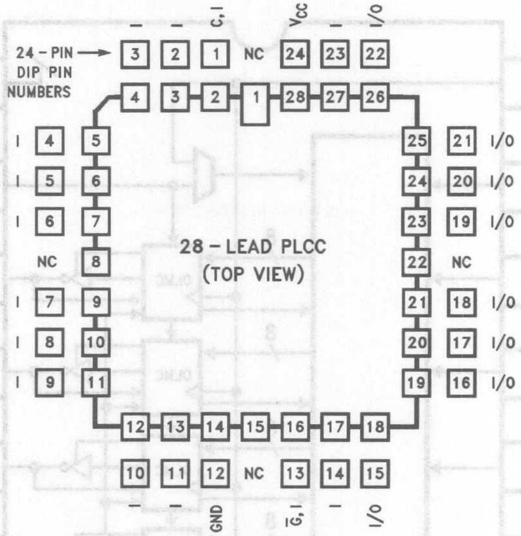
GAL20V8 Block Diagram—DIP Connections



TL/L/11256-10

FIGURE 1

28-Lead PLCC Connection Diagram



TL/L/11256-11

FIGURE 2

Clock/Input Frequency Specifications

The clock frequency (f_{CLK}) parameter listed in the Recommended Operating Conditions table specifies the maximum speed at which the GAL registers are guaranteed to operate. Clock frequency is defined differently for the two cases in which register feedback is used versus when it is not. In a data-path type application, when the logic functions fed into the registers are not dependent on register feedback from the previous cycle (i.e., based only on external inputs), the minimum required cycle period (f_{CLK}^{-1} without feedback) is defined as the greater of the minimum clock period (t_w high + t_w low) and the minimum "data window" period (t_{SU} + t_H). This assumes optimal alignment between data inputs and the clock input. In sequential logic applications such as state machines, the minimum required cycle period ($t_{CYCLE} = f_{CLK}^{-1}$ with feedback) is defined as $t_{CLK} + t_{SU}$. This provides sufficient time for outputs from the registers to feed back through the logic array and set up on the inputs to the registers before the end of each cycle.

The input frequency (f_I) parameter specifies the maximum rate at which each GAL input can be toggled and still produce valid logic transitions on each combinatorial output. The f_I specification is derived as the inverse of the combinatorial propagation delay (t_{PD}).

Design Development Support

A variety of software tools and programming equipment is available to support the development of designs using GAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate a JEDEC-compatible "cell-map" (analogous to a PAL "fuse-map"). The industry-standard JEDEC format ensures that the resulting cell-map file can be down-loaded into a variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible.

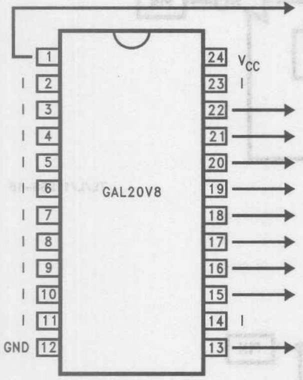
National strongly recommends using only approved programming hardware and software for developing GAL designs. Programming using unapproved equipment generally voids all guarantees. Approved programmers incorporate specialized programming algorithms that program the array and automatically configure the architecture cells. To ensure data retention and reliability, the programming algorithm also tracks the number of programming cycles to which each GAL device has been subjected since shipment, and stores this information automatically in the device.

Design Development Support (Continued)

The special GAL programming algorithm can also program a GAL device using a standard fuse-map developed for any of the emulated PAL products. PAL fuse-maps can be created by any JEDEC-compatible PAL development software or by loading the fuse pattern from an existing programmed PAL device into the programming unit (provided the PAL device has not been secured). However, to utilize the full flexibility of the GAL architecture, true GAL development software (such as OPAL software) is recommended.

Detailed logic diagrams showing all JEDEC cell-map addresses in the GAL logic array and OLMC are provided for direct map editing and diagnostic purposes (see "Programming Details"). For a list of current software and programming support tools available for these devices, please contact your local National sales representative or distributor. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

OLMC Selection Table



TL/L/11256-12

TABLE I

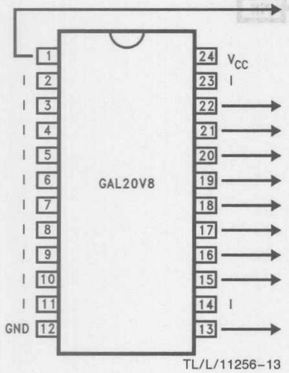
"Small-PAL" Mode	"Registered-PAL" Mode	"Medium-PAL" Mode
INPUT	CLOCK	INPUT
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT	OUTPUT ENABLE ($\overline{G}$)	INPUT

*Active combinatorial output

**TRI-STATE combinatorial output

Note: Pin numbers above apply to 24-pin DIP packages; refer to the 28-lead PCC Connection Diagram for conversion.

PAL Replacement Configurations



TL/L/11256-13

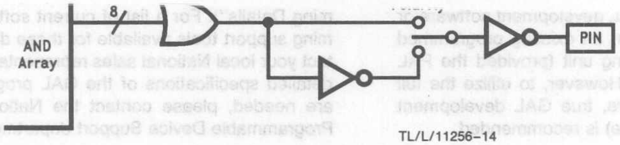
TABLE II

	"Small-PAL" Mode				"Registered-PAL" Mode			"Medium-PAL" Mode
	INPUT	INPUT	INPUT	INPUT	CLOCK	CLOCK	CLOCK	INPUT
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	INPUT	INPUT	INPUT	INPUT	$\overline{G}$	$\overline{G}$	$\overline{G}$	INPUT
Emulated PAL Products	14L8 14H8 14P8	16L6 16H6 16P6	18L4 18H4 18P4	20L2 20H2 20P2	20R8 20RP8	20R6 20RP6	20R4 20RP4	20L8 20H8 20P8

* Active combinatorial output.

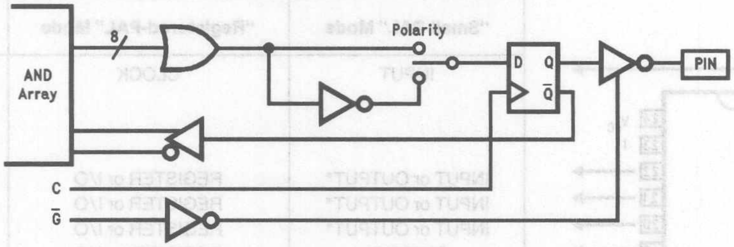
**TRI-STATE combinatorial output.

Note: Pin numbers above apply to 24-pin DIP packages; refer to the 28-pin PCC Connection Diagram for conversion.



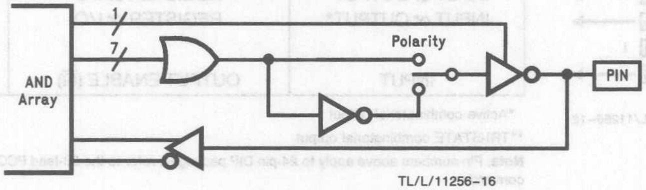
TL/L/11256-14

REGISTER (Registered Output)



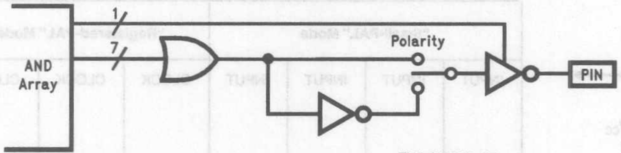
TL/L/11256-15

I/O (Combinatorial Input/Output)



TL/L/11256-16

TRI-STATE (TRI-STATE Combinatorial Output)



TL/L/11256-17

FIGURE 3

cess is disabled, preventing further programming or verification of the array. The security cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed.

Electronic Signature

Each GAL device contains an electronic signature word consisting of 64 bits of reprogrammable memory. The electronic signature word can be programmed to contain any identification information desired by the user. Some uses include pattern identification labels, revision numbers, dates, inventory control information, etc. The data stored in the electronic signature word has no effect on the functionality of the device. The information is read out of the device using the normal program verification procedure provided by the programming equipment. The information may be accessed at any time independent of the state of the security cell. National's OPAL development software allows electronic signature data to be entered by the user and downloaded to the programming equipment.

Bulk Erase

The programming equipment automatically performs a bulk erase operation prior to each programming operation. No special erase operation need be performed by the user. Bulk erase clears the logic array, architecture cells, security cell, and electronic signature information. The GAL device is thereby reverted back to its virgin state.

Latch-Up Protection

GAL devices are designed with an on-chip charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

To insure that no undesired bias conditions occur with P+ diffusions, a Latch-Lock™ power-up circuitry has been developed. The drain of all P channel devices normally connected to the device supply are now connected to an alternate supply that powers up after the device N-wells have been biased and the substrate has reached its negative clamp value. This prevents any hazardous bias conditions from developing in the power-up sequence. After power-up is complete, the Latch-Lock circuitry becomes dormant until a full power-down has occurred; this eliminates the chance of an unwanted P channel power-down during device operation.

Manufacturer Testing

Because of EECMOS technology, GAL devices can be reprogrammed in milliseconds. This allows each device to be completely tested by the manufacturer using numerous logic array and architecture patterns prior to shipping. Every programmable cell and every logic path through every de-

specifications.

The testing procedure performed on all GAL devices by the manufacturer tests all aspects of device operation. Extensive testing of all programmable cells in the device include margin testing, internal verify, and program retention during high-temperature bake. All DC and AC parameters are tested at hot and cold temperatures using a variety of worst-case logic and signal patterns. Functional tests include reprogramming each OLMC to all valid architectural configurations.

Register Preload

The register preload feature allows OLMC registers to be directly loaded with any desired data pattern. It also allows the present state of OLMC registers to be examined regardless of TRI-STATE control conditions. This simplifies testing of devices after programming. A device may be put into any desired register state at any point during the functional test sequence. The test sequence may then be resumed to verify proper next-state transitions. This allows complete verification of sequential logic circuits, including states that are normally impossible or difficult to reach. It may also shorten the overall test time significantly.

Register preload is not an operational mode and is not intended for board-level testing because elevated voltage levels must be applied to the device. The programming equipment normally provides the register preload capability as part of its functional test facility. Note that the testing of GAL devices after programming by the user may be considered unnecessary because all EECMOS GAL products are completely tested by the manufacturer, guaranteeing 100% post-programming functional yield.

The register preload algorithm is described for those users who wish to test programmed GAL devices using test equipment other than approved GAL programming equipment. As shown in the Register Preload Waveform in *Figure 5,* the preload sequence must not begin until the normal power-up reset operation has completed (after time t_{RESET}). The device is placed into preload mode by raising the "PRLD" input (pin* 13) to voltage V_{IES}, as specified in the Register Preload Specifications (Table III).

To preload the OLMC registers, a series of data bits are shifted into the device on the "S_{DIN}" input (pin* 11), one bit for each OLMC in which registered output has been selected. (Non-registered OLMCs are bypassed.) The shift sequence is clocked by the rising edge of the "D_{CLK}" input (pin* 1). The data stream is shifted in through the registered OLMC with the lowest corresponding pin number, and then "upward" through all remaining registered OLMCs in pin-number ascending order. Therefore, the first data bit in the series is ultimately loaded into the registered OLMC with the highest corresponding pin number, as shown in *Figure 4.*

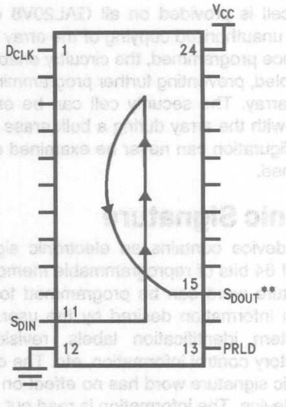
*Applies to 24-pin DIP packages for GAL20V8; refer to the 28-lead PLCC Connection Diagram for conversion.

Register Preload (Continued)

As the data series is shifted into the S_{DIN} input, the contents of all registers (in registered OLMCs) are shifted "upward" and out onto the "S_{DOUT}" output (pin* 15). Complete present-state information can be examined in this manner. Test fixtures can be devised to test several GAL devices in which the S_{DOUT} pin of each chip is connected to the S_{DIN} pin of the next, and all preload and present-state data can be shifted around a single serial loop.

Note that when shifting register data into S_{DIN} or out of S_{DOUT}, V_{IL}/V_{OL} = register reset (0), and V_{IH}/V_{OH} = register set (1). These 0 and 1 register states are always inverted (active-low) on the normal output pins regardless of the selected output polarity (polarity affects logic function values before register inputs).

*Applies to 24-pin DIP packages for GAL20V8; refer to the 28-lead PLCC Connection Diagram for conversion.



TL/L/11256–18

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

FIGURE 4. Output Register Preload Pinout

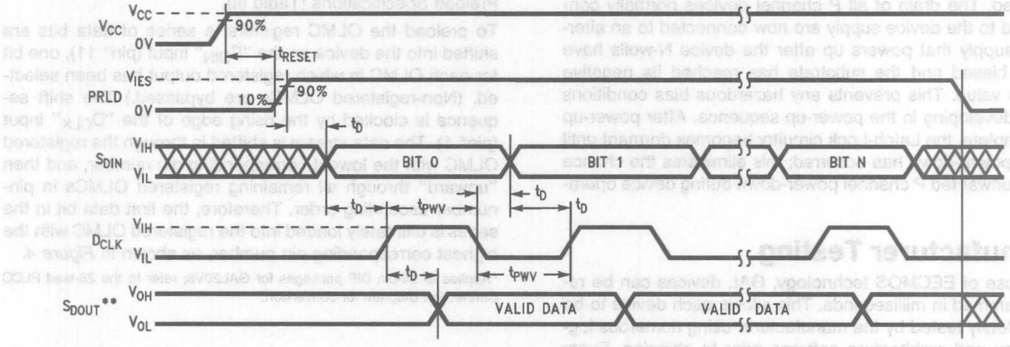
Register Preload Specifications

TABLE III

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input Voltage (High)		2.40		V_{CC}	V
V_{IL}	Input Voltage (Low)		0.00		0.50	V
V_{IES}	Register Preload Input Voltage		11.5	12	12.5	V
V_{OH}	Output Voltage (High) (Note 1)				V_{CC}	V
V_{OL}	Output Voltage (Low) (Note 1)	$I_{OL} \leq 12$ mA	0.00		0.50	V
I_{IH}, I_{IL}	Input Current (Programming)			±1	±10	μA
I_{OH}	High Level Output Current (Note 1)	$V_{OH} \leq V_{CC}$			10	μA
t_{PWV}	Verify Pulse Width		1	5	10	μs
t_D	Pulse Sequence Delay		1	5	10	μs
t_{RESET}	Register Reset Time from Valid V_{CC}				45	μs

Note 1: The S_{DOUT} output buffer is an open drain output. This pin should be terminated to V_{CC} with a 10k resistor.

Register Preload Waveforms

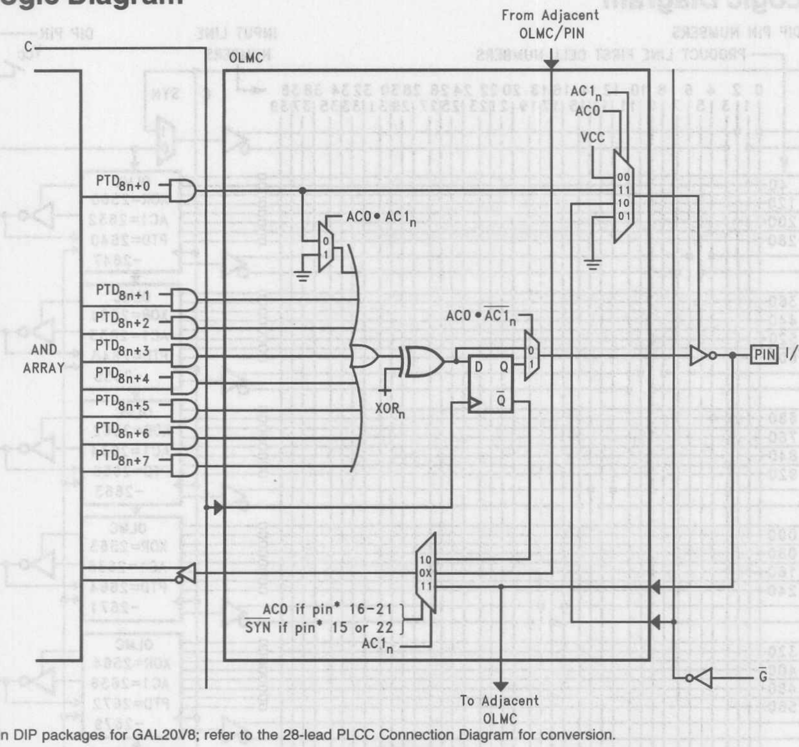


TL/L/11256–19

FIGURE 5

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

OLMC Logic Diagram



*Applies to 24-pin DIP packages for GAL20V8; refer to the 28-lead PLCC Connection Diagram for conversion.

TL/L/11256–20

FIGURE 6

OLMC Architecture Programming

TABLE IV

	"Small-PAL" Mode			"Registered-PAL" Mode			"Medium-PAL" Mode	
	Function		JEDEC Input Line #s (Note 1)	Function		JEDEC Input Line #s (Note 1)	Function	JEDEC Input Line #s (Note 1)
Pin 1	INPUT	INPUT	2, 3	CLOCK	CLOCK		INPUT	2, 3
Pin 23	INPUT	INPUT	6, 7	INPUT	INPUT	2, 3	INPUT	6, 7
***Pin 22	I/O	INPUT	10, 11	REGISTER	I/O	6, 7	TRI-STATE**	
***Pin 21	I/O	INPUT	14, 15	REGISTER	I/O	10, 11	I/O	10, 11
***Pin 20	I/O	INPUT	18, 19	REGISTER	I/O	14, 15	I/O	14, 15
***Pin 19	OUTPUT*	NC		REGISTER	I/O	18, 19	I/O	18, 19
***Pin 18	OUTPUT*	NC		REGISTER	I/O	22, 23	I/O	22, 23
***Pin 17	I/O	INPUT	22, 23	REGISTER	I/O	26, 27	I/O	26, 27
***Pin 16	I/O	INPUT	26, 27	REGISTER	I/O	30, 31	I/O	30, 31
***Pin 15	I/O	INPUT	30, 31	REGISTER	I/O	34, 35	TRI-STATE**	
Pin 14	INPUT	INPUT	34, 35	INPUT	INPUT	38, 39	INPUT	34, 35
Pin 13	INPUT	INPUT	38, 39	$\overline{G}$	$\overline{G}$		INPUT	38, 39
	AC1$_n$ = 0	AC1$_n$ = 1		AC1$_n$ = 0	AC1$_n$ = 1		AC1$_n$ 1	
	SYN = 1, AC0 = 0			SYN = 0, AC0 = 1			SYN = 1, AC0 = 1	
	All outputs are combinatorial and always active.			At least one output is registered.			All I/O pins are combinatorial.	

Note: Pin numbers above apply to 24-pin DIP packages; refer to the 28-lead PLCC Connection Diagram for conversion.

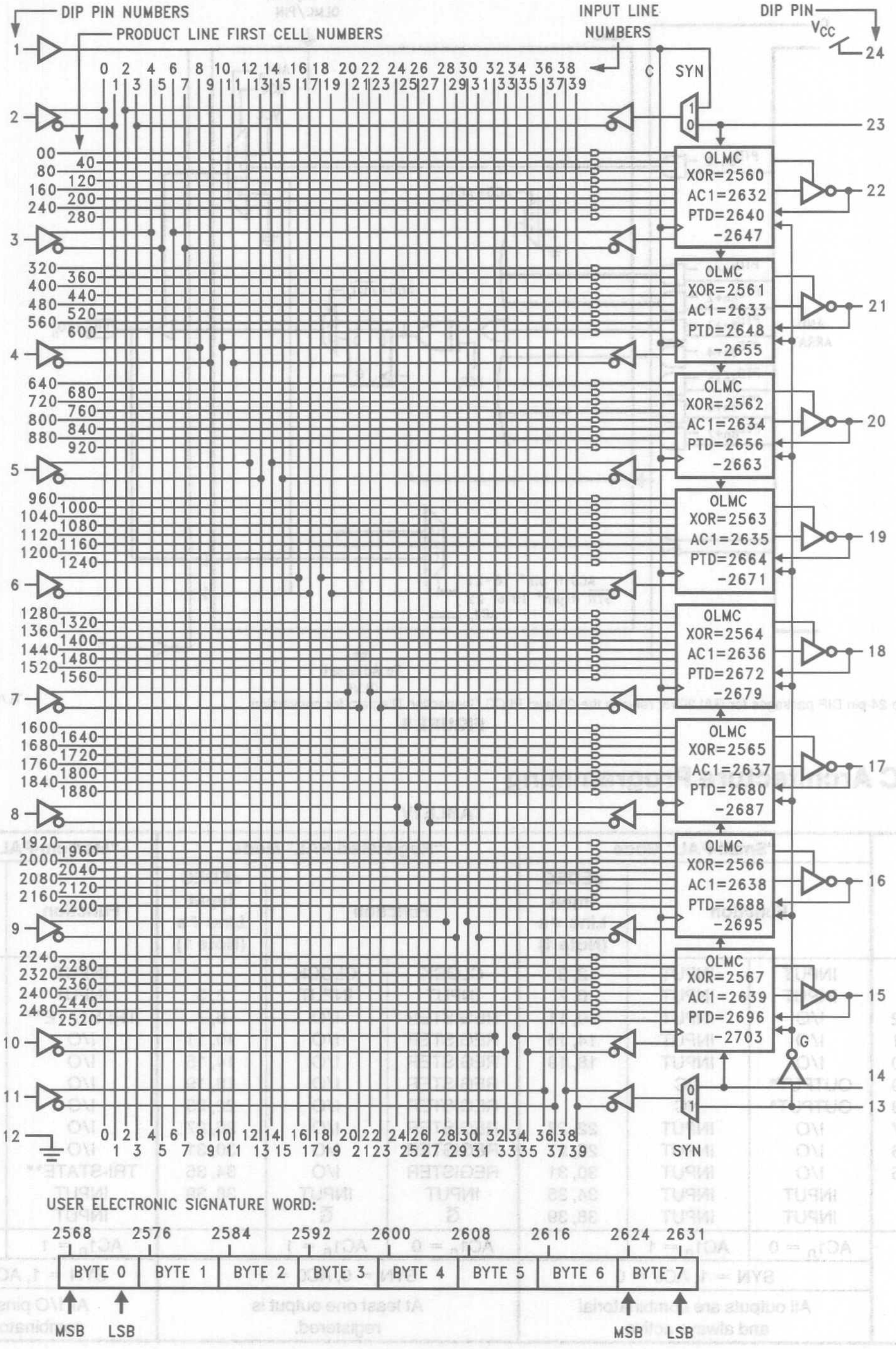
Note 1: All even and odd numbered JEDEC input line numbers correspond to true and complement array inputs, respectively.

*Active combinatorial output.

**TRI-STATE combinatorial output.

***AC1$_n$ applies to these I/O pins only.

GAL20V8 Logic Diagram



SYN=2704
AC0=2705

JEDEC Logic Array Cell Number = Product Line First Cell Number + Input Line Number

TL/L/11256-21

FIGURE 7

Programming Details

Understanding the information in this section is not essential when using approved programming equipment and software for developing GAL designs. This is a more thorough disclosure of the GAL architecture provided for direct JEDEC cell-map editing and diagnostic purposes. This section alone, however, does not contain sufficient information to implement the GAL programming algorithm. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

As mentioned in the Functional Description, the OLMC is responsible for selecting input and/or output paths, registered vs. combinatorial outputs, active-high or low polarity, and common vs. locally-controlled TRI-STATE control. Additionally, the OLMCs select between alternate logic array input paths to maintain JEDEC cell-map compatibility with either "small-PAL" or "medium-PAL" logic arrays.

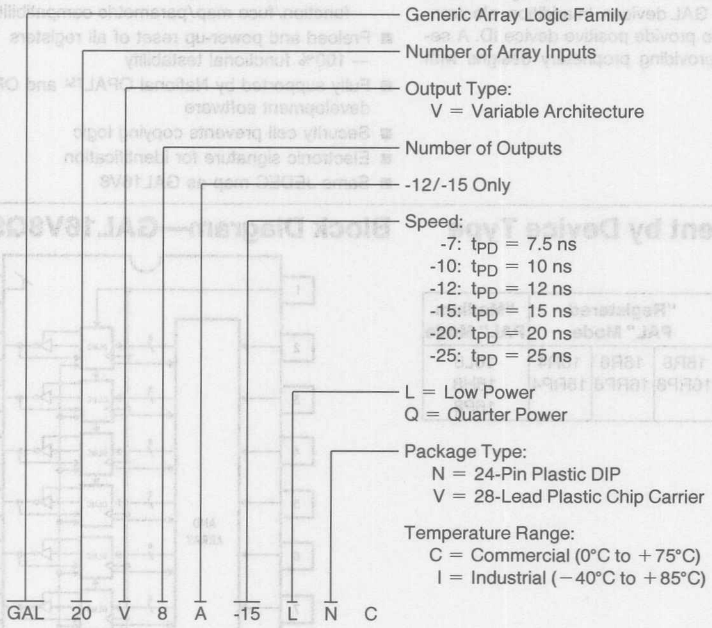
The various configurations of the OLMCs are controlled by a set of programmable "architecture" cells, separate from the logic-defining array cells. Each GAL device contains two "global" architecture cells, "SYN" and "AC0", which affect all OLMCs. Each of the devices's eight OLMCs also contains two "local" cells, "AC1" and "XOR". The OLMC Logic Diagram in *Figure 6* shows how the architecture cells select the different paths through the OLMC.

The SYN bit controls whether a device will have any registered outputs (SYN = 0) or will be purely combinatorial (SYN = 1). The SYN bit determines whether device pins* 1 and 13 are used as the clock and global TRI-STATE control inputs (SYN = 0) or whether they are ordinary inputs (SYN = 1). The AC0 bit selects between the "Small-PAL" mode and the "Medium/Registered-PAL" modes. The function of the AC1 bits depend on the state of the AC0 bit. In "Small-PAL" mode (AC0 = 0), the AC1 bit in each OLMC determines whether the associated device pin is an output (AC1 = 0) or an input (AC1 = 1). In "Registered-PAL" mode (AC0 = 1), the AC1 bit determines whether each OLMC is registered (AC1 = 0) or combinatorial (AC1 = 1). In "Medium-PAL" mode (AC0 = 1), the AC1 bits in all OLMCs must be set to 1 (combinatorial). All of the valid architecture bit configurations are shown in the OLMC Architecture table (Table IV), which has the same familiar format used in the OLMC Selection table (Table I).

Independent of SYN, AC0 and the AC1 bits, the XOR bit in each OLMC selects between active-low (XOR = 0) or active-high (XOR = 1) output polarity.

*Applies to 24-pin DIP packages for GAL20V8; refer to the 28-lead PLCC Connection Diagram for conversion.

Ordering Information



Generic Array Logic Family

Number of Array Inputs

Output Type:
V = Variable Architecture

Number of Outputs

-12/-15 Only

Speed:
-7: t_{PD} = 7.5 ns
-10: t_{PD} = 10 ns
-12: t_{PD} = 12 ns
-15: t_{PD} = 15 ns
-20: t_{PD} = 20 ns
-25: t_{PD} = 25 ns

L = Low Power
Q = Quarter Power

Package Type:
N = 24-Pin Plastic DIP
V = 28-Lead Plastic Chip Carrier

Temperature Range:
C = Commercial (0°C to +75°C)
I = Industrial (−40°C to +85°C)

THE GAL20V8A-10L HAS BEEN RENAMED GAL20V8-10L.
THERE WERE NO SPECIFICATION CHANGES ASSOCIATED WITH THIS NAME CHANGE.

GAL16V8QS 20-Pin Generic Array Logic Family

General Description

The EECMOS GAL® QS™ devices combine a high performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels. The GALQS family utilizes NSC Quiet Series technology to guarantee quiet output switching and improve dynamic threshold performance. GAL Quiet Series™ features GTO™ output control in addition to a split ground bus for superior performance.

The 20-pin GAL16V8QS features 8 programmable Output Logic Macrocells (OLMCs) allowing each TRI-STATE® output to be configured by the user. Additionally, the GAL16V8QS is capable of emulating, in a functional/fuse map/parametric compatible device, all common 20-pin PAL® device architectures.

Programming is accomplished using readily available hardware and software tools. NSC guarantees a minimum 100 erase/write cycles.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, NSC guarantees 100% field programmability and functionality of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

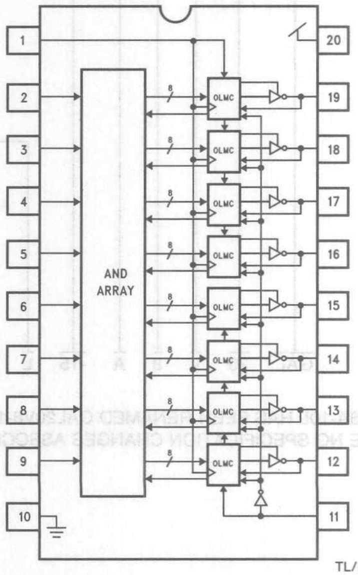
Features

- High performance EECMOS technology
  - — 15 ns maximum propagation delay
  - — f_{CLK} = 62.5 MHz
  - — 10 ns maximum from clock input to data output
  - — TTL compatible 24 mA outputs
  - — UltraMOS® III advanced CMOS technology
- Guaranteed simultaneous switching noise level and dynamic threshold performance
- Reduced groundbounce and undershoot
- Reduced power
  - — Low power = 90 mA I_{CC} max
  - — Quarter power = 55 mA max I_{CC} max
- Electrically erasable cell technology
  - — Reconfigurable logic
  - — Reprogrammable cells
  - — 100% tested/guaranteed 100% yields
  - — High speed electrical erasure (<50 ms)
  - — 20 year data retention
- Eight output logic macrocells
  - — Maximum flexibility for complex logic designs
  - — Programmable output polarity
  - — Also emulates 20-pin PAL devices with full function/fuse map/parametric compatibility
- Preload and power-up reset of all registers
  - — 100% functional testability
- Fully supported by National OPAL™ and OPALjr development software
- Security cell prevents copying logic
- Electronic signature for identification
- Same JEDEC map as GAL16V8

PAL Replacement by Device Type

"Small PAL" Mode				"Registered PAL" Mode			"Medium PAL" Mode
10L8	12L6	14L4	16L2	16R8	16R6	16R4	16L8
10H8	12H6	14H4	16H2	16RP8	16RP6	16RP4	16H8
10P8	12P6	14P4	16P2				16P8

Block Diagram—GAL16V8QS



TL/L/11145–1

GAL16V8QS (-15L, -15Q, -20L, -25L, -25Q) Commercial

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage V_{CC}	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C

Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-026 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Units
		Min	Nom	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	V
T_A	Operating Free-Air Temperature	0	25	75	°C
T_C	Operating Case Temperature				°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL16V8QS-15L		GAL16V8QS-15Q		GAL16V8QS-20L		GAL16V8QS-25L/25Q		Units
			COM		COM		COM		COM		
			Min	Max	Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		12		12		15		15		ns
t_H	Hold Time (Input after Clock)		0		0		0		0		ns
t_W	Clock Pulse Width (High/Low)		8		8		12		12		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		22		22		27		27		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		45.5		45.5		37		37	MHz
		Without Feedback		62.5		62.5		41.66		41.66	
f_I	Input Frequency (Note 5)			66.6		66.6		50		50	
t_{PR}	Clock Valid after Power-Up			100		100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) = $(t_{CYCLE})^{-1}$
f_{CLK} (without feedback) = $(2\ t_W)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL16V8QS (-15L, -15Q, -20L, -25L, -25Q) Commercial (Continued)

Quiet Electrical Characteristics (V_{CC} = 5.0V, Temp = 25°C)

Symbol	Parameter	Conditions	Typ	Max	Units
V_{OLP}	Quiet Output Maximum Dynamic V_{OL}	(Note 6)	1.1	1.5	V
V_{OLV}	Quiet Output Minimum Dynamic V_{OL}	(Note 6)	−0.6	−1.2	V
V_{IHD}	Maximum High Level Dynamic Input Voltage	(Note 7)	1.9	2.2	V
V_{ILD}	Maximum Low Level Dynamic Input Voltage	(Note 7)	1.2	0.8	V

Note 6: 7 outputs switching from high to low, one output at low. The width of the bounce at 50% amplitude is ≤3 ns.

Note 7: 7 outputs switching. Input-under-test switching:
3V to threshold (V_{ILD})
0V to threshold (V_{IHD})
f = 1 MHz

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		V_{CC}+1	V
V_{IL}	Low Level Input Voltage				−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = 3.2 mA	COM	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 24 mA	COM			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC}(Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC}(Max)					10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC}(Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	-15L, -20L, -25L	COM			90	mA
			-15Q, -25Q	COM			55	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					10	pF

*One output at a time for a maximum duration of one second.

GAL16V8QS(-15L, -15Q, -20L, -25L, -25Q) Commercial (Continued)

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL16V8QS-15L/-15Q COM		GAL16V8QS-20L COM		GAL16V8QS-25L/25Q COM		Units
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, $C_L = 50$ pF		15		20		25	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, $C_L = 50$ pF		10		12		12	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High: S1 Open, $C_L = 50$ pF Active Low: S1 Closed, $C_L = 50$ pF		15		18		20	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}: S1 Open, $C_L = 5$ pF From V_{OL}: S1 Closed, $C_L = 5$ pF		15		18		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High: S1 Open, $C_L = 50$ pF Active Low: S1 Closed, $C_L = 50$ pF		15		20		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}: S1 Open, $C_L = 5$ pF From V_{OL}: S1 Closed, $C_L = 5$ pF		15		20		25	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, $C_L = 50$ pF		45		45		45	µs

GAL16V8QS(-20L, -25L, -25Q) Industrial

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage V_{CC}	$-0.5V$ to $+7.0V$
Input Voltage (Note 2)	$-2.5V$ to V_{CC} $+1.0V$
Off-State Output Voltage (Note 2)	$-2.5V$ to V_{CC} $+1.0V$
Output Current	±100 mA
Storage Temperature	$-65°C$ to $+150°C$

Ambient Temperature with Power Applied	$-65°C$ to $+125°C$
Junction Temperature	$-65°C$ to $+150°C$
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	1000V

$C_{ZAP} = 100$ pF
$R_{ZAP} = 1500\Omega$
Test Method: Human Body Model
Test Specification: NSC SOP-5-026 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Industrial			Units
		Min	Nom	Max	
V_{CC}	Supply Voltage	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	−40	25	85	°C
T_C	Operating Case Temperature				°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL16V8QS-20L		GAL16V8QS-25L		GAL16V8QS-25Q		Units
			IND		IND		IND		
			Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		15		20		20		ns
t_H	Hold Time (Input after Clock)		0		0		0		ns
t_W	Clock Pulse Width (High/Low)		12		15		15		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		30		35		35		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		33.3		28.5		28.5	MHz
		Without Feedback		41.66		33.3		33.3	
f_I	Input Frequency (Note 5)			50		40		40	
t_{PR}	Clock Valid after Power-Up			100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) $= (t_{CYCLE})^{-1}$
f_{CLK} (without feedback) $= (2\ t_W)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL16V8QS (-20L, -25L, -25Q) Industrial (Continued)

Quiet Electrical Characteristics (V_{CC} = 5.0V, Temp = 25°C)

Symbol	Parameter	Conditions	Typ	Max	Units
V_{OLP}	Quiet Output Maximum Dynamic V_{OL}	(Note 6)	1.1	1.5	V
V_{OLV}	Quiet Output Minimum Dynamic V_{OL}	(Note 6)	−0.6	−1.2	V
V_{IHD}	Maximum High Level Dynamic Input Voltage	(Note 7)	1.9	2.2	V
V_{ILD}	Maximum Low Level Dynamic Input Voltage	(Note 7)	1.2	0.8	V

Note 6: 7 outputs switching from high to low, one output at low. The width of the bounce at 50% amplitude is ≤3 ns.

Note 7: 7 outputs switching. Input-under-test switching:
3V to threshold (V_{ILD})
0V to threshold (V_{IHD})
f = 1 MHz

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		$V_{CC}+1$	V
V_{IL}	Low Level Input Voltage				−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA	IND	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 24 mA	IND			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC}(Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC}(Max)					10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC}(Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	-20L/-25L	IND			90	mA
			-25Q	IND			55	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					10	pF

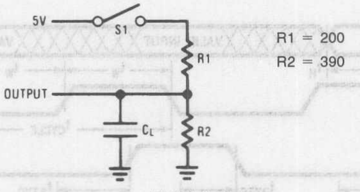
*One output at a time for a maximum duration of one second.

GAL16V8QS (-20L, -25L, -25Q) Industrial (Continued)

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL16V8QS-20L		GAL16V8QS-25L		GAL16V8QS-25Q		Units
			IND		IND		IND		
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, $C_L = 50$ pF		20		25		25	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, $C_L = 50$ pF		15		15		15	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High: S1 Open, $C_L = 50$ pF Active Low: S1 Closed, $C_L = 50$ pF		18		20		20	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}: S1 Open, $C_L = 5$ pF From V_{OL}: S1 Closed, $C_L = 5$ pF		18		20		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High: S1 Open, $C_L = 50$ pF Active Low: S1 Closed, $C_L = 50$ pF		20		25		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}: S1 Open, $C_L = 5$ pF From V_{OL}: S1 Closed, $C_L = 5$ pF		20		25		25	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, $C_L = 50$ pF		45		45		45	µs

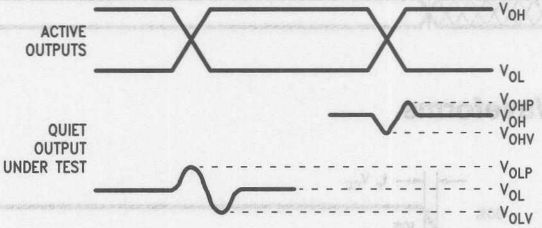
AC Test Load



TL/L/11145-2

Test Waveforms

Quiet Output Noise Voltage Waveforms



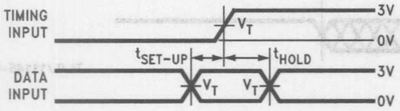
TL/L/11145-24

Note A. V_{OHV} and V_{OLP} are measured with respect to ground reference.

Note B. Input pulses have the following characteristics: f = 1 MHz, t_r = 3 ns, t_f = 3 ns, skew < 150 ps.

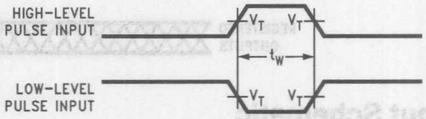
Note C. Test load for Quiet output: C_L = 50 pF, R_L = 500Ω.

Setup and Hold



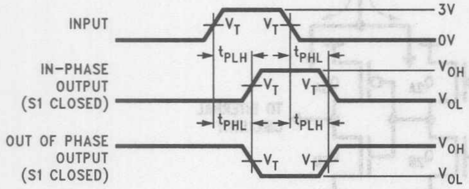
TL/L/11145-3

Pulse Width



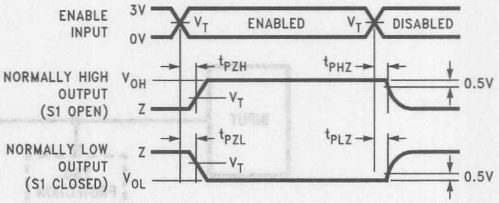
TL/L/11145-4

Propagation Delay



TL/L/11145-5

Enable and Disable



TL/L/11145-6

Notes:

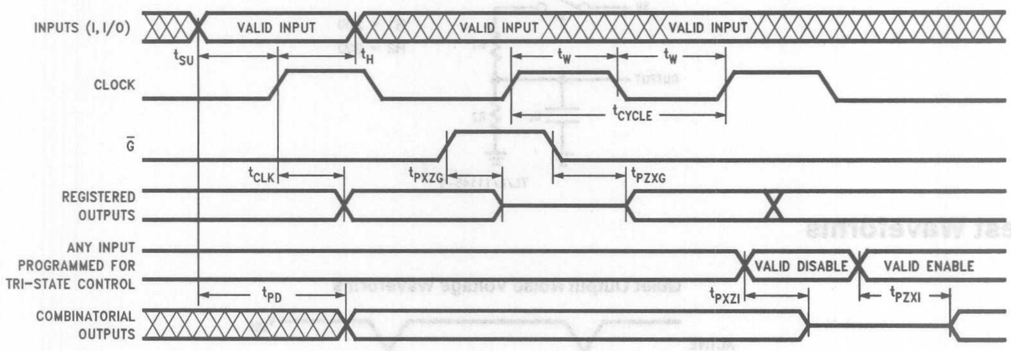
C_L includes probe and jig capacitance.

V_T = 1.5V.

Test inputs have rise and fall times of 5 ns between 0.3V and 2.7V.

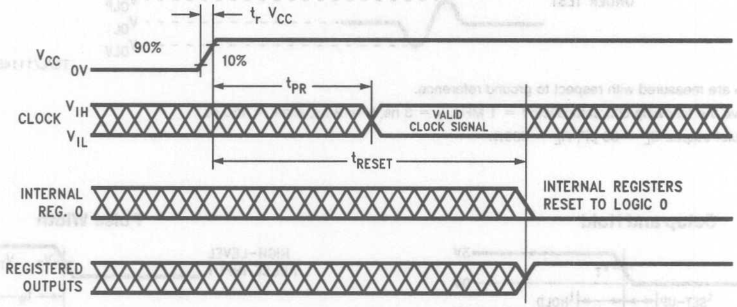
In the examples above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Switching Waveforms



TL/L/11145-7

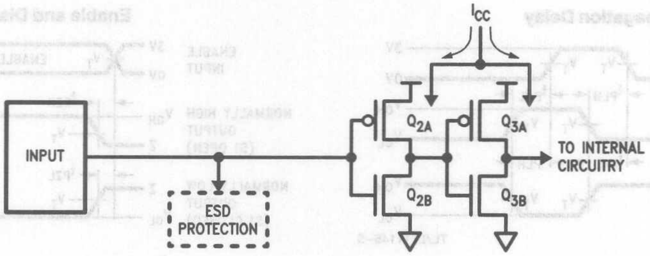
Power-Up Reset Waveforms



TL/L/11145-8

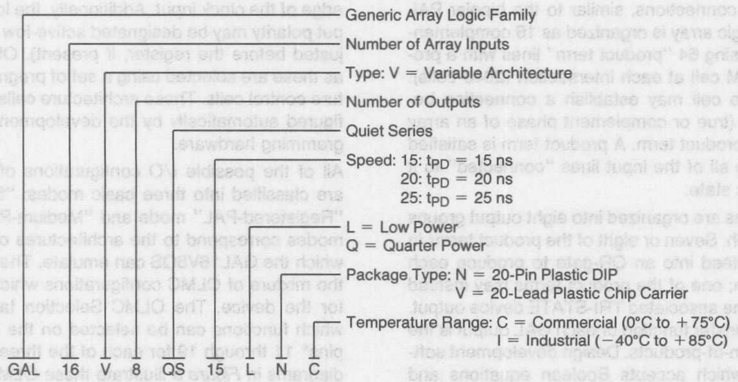
Input Schematic

Input Translator/Buffer



TL/L/11145-9

Ordering Information



Generic Array Logic Family

Number of Array Inputs

Type: V = Variable Architecture

Number of Outputs

Quiet Series

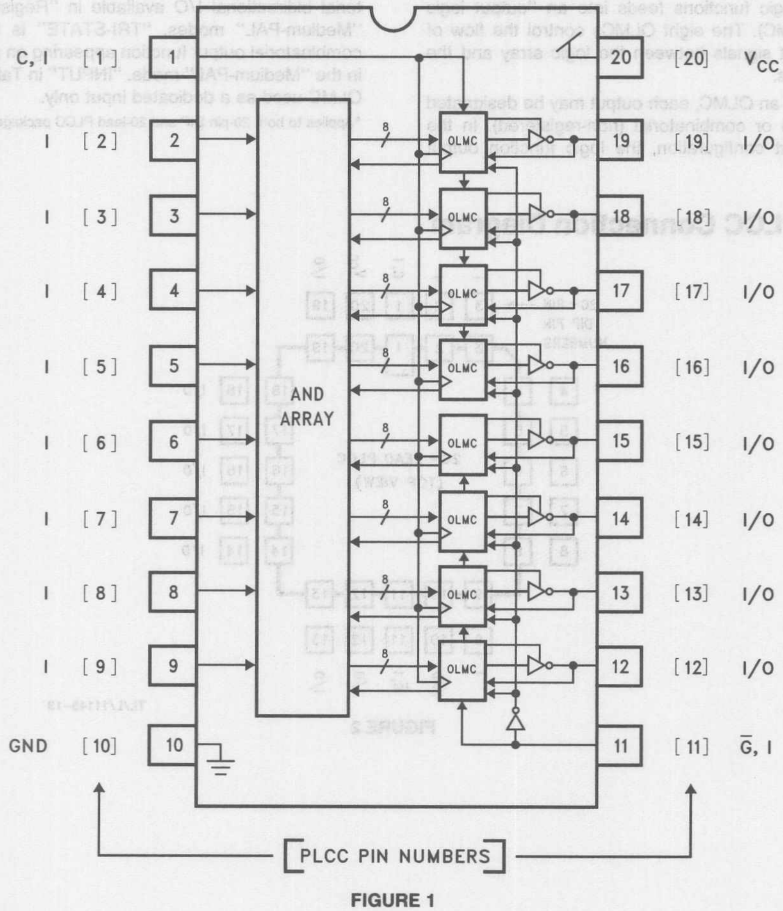
Speed: 15: t_{PD} = 15 ns
20: t_{PD} = 20 ns
25: t_{PD} = 25 ns

L = Low Power
Q = Quarter Power

Package Type: N = 20-Pin Plastic DIP
V = 20-Lead Plastic Chip Carrier

Temperature Range: C = Commercial (0°C to +75°C)
I = Industrial (−40°C to +85°C)

GAL16V8QS Block Diagram—DIP Connections



TL/L/11145-12

FIGURE 1

Functional Description

The GAL logic array consists of a programmable AND array with fixed OR-gate connections, similar to the bipolar PAL architecture. The logic array is organized as 16 complementary input lines crossing 64 "product term" lines with a programmable E^{2}PROM cell at each intersection (2048 cells). Each programmable cell may establish a connection between an input line (true or complement phase of an array input signal) and a product term. A product term is satisfied (logically true) while all of the input lines "connected" to it are in the high logic state.

The 64 product terms are organized into eight output groups with eight terms each. Seven or eight of the product terms in each output group feed into an OR-gate to produce each output logic function; one of the product terms may instead be used to control the associated TRI-STATE device output. The fundamental transfer function of each GAL output is the familiar Boolean sum-of-products. Design development software is available which accepts Boolean equations and converts them automatically into GAL programming patterns.

As shown in the GAL16V8QS Block Diagram *(Figure 1)*, a total of eight output logic functions are available. Each of the AND/OR logic functions feeds into an "output logic macrocell" (OLMC). The eight OLMCs control the flow of input and output signals between the logic array and the device's I/O pins.

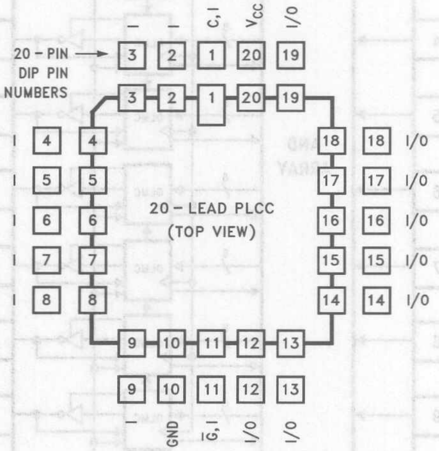
Under control of an OLMC, each output may be designated either registered or combinatorial (non-registered). In the registered output configuration, the logic function output passes through a D-type flip-flop triggered by the rising edge of the clock input. Additionally, the logic function's output polarity may be designated active-low or active-high (adjusted before the register, if present). OLMC options such as these are selected using a set of programmable architecture control cells. These architecture cells are normally configured automatically by the development software or programming hardware.

All of the possible I/O configurations of the GAL16V8QS are classified into three basic modes: "Small-PAL" mode, "Registered-PAL" mode and "Medium-PAL" mode. These modes correspond to the architectures of the PAL families which the GAL16V8QS can emulate. The modes determine the mixture of OLMC configurations which can be selected for the device. The OLMC Selection table (Table I) lists which functions can be selected on the device pin* 1 and pins* 11 through 19 for each of the three modes. The logic diagrams in *Figure 3* illustrate these OLMC functions.

"OUTPUT" represents the always-active combinatorial output configuration available in the "Small-PAL" mode. "REGISTER" is the registered output with register feedback available in the "Registered-PAL" mode. "I/O" is the combinatorial bidirectional I/O available in "Registered-PAL" and "Medium-PAL" modes. "TRI-STATE" is the TRI-STATE combinatorial output function appearing on pins* 12 and 19 in the "Medium-PAL" mode. "INPUT" in Table I denotes an OLMC used as a dedicated input only.

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8QS.

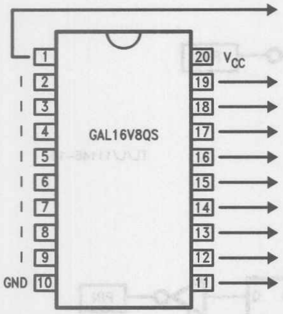
20-Lead PLCC Connection Diagram



TL/L/11145-13

FIGURE 2

OLMC Selection Table



TL/L/11145-14

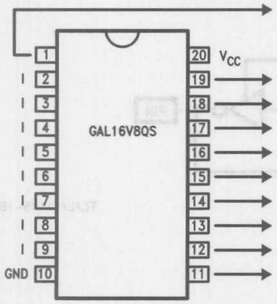
TABLE I

"Small-PAL" Mode	"Registered-PAL" Mode	"Medium-PAL" Mode
INPUT	CLOCK	INPUT
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT	OUPUT ENABLE ($\overline{G}$)	INPUT

*Active combinatorial output

**TRI-STATE combinatorial output

PAL Replacement Configurations



TL/L/11145-15

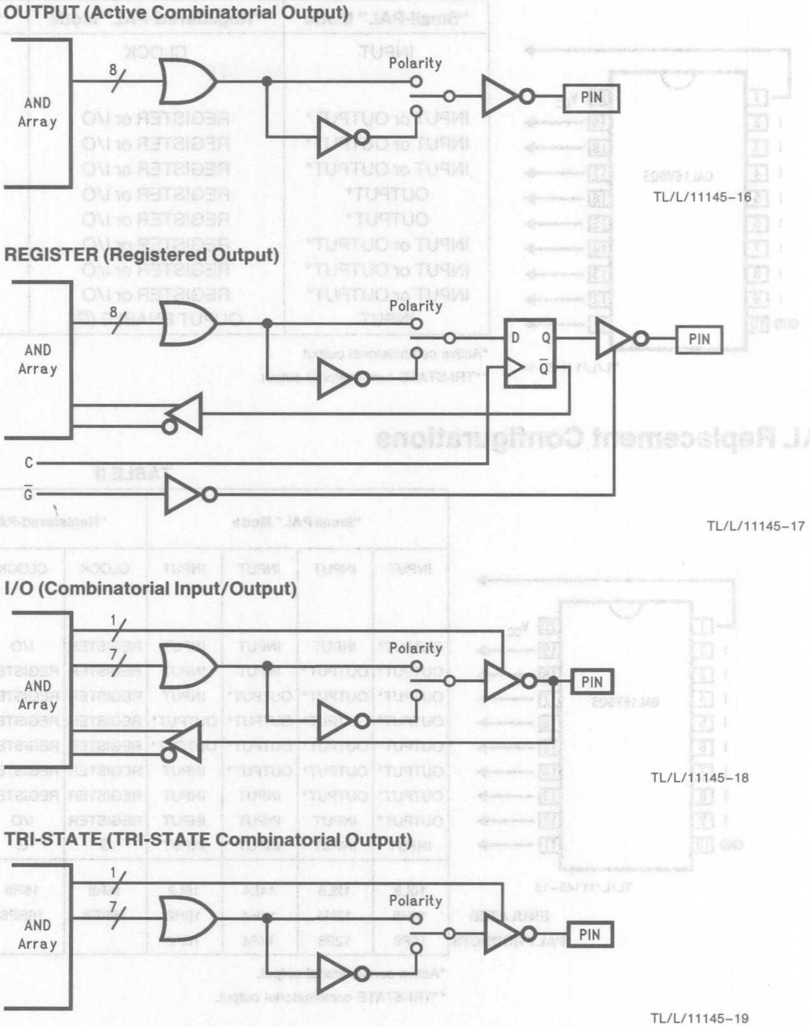
TABLE II

	"Small PAL" Mode				"Registered-PAL" Mode			"Medium-PAL" Mode
	INPUT	INPUT	INPUT	INPUT	CLOCK	CLOCK	CLOCK	INPUT
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	INPUT	INPUT	INPUT	INPUT	$\overline{G}$	$\overline{G}$	$\overline{G}$	INPUT
	10L8	12L6	14L4	16L2	16R8	16R6	16R4	16L8
EMULATED	10H8	12H6	14H4	16H2	16RP8	16RP6	16RP4	16H8
PAL PRODUCTS	10P8	12P6	14P4	16P2				16P8

*Active combinatorial output.

**TRI-STATE combinatorial output.

OLMC Configurations



OUTPUT (Active Combinatorial Output)

TL/L/11145–16

REGISTER (Registered Output)

TL/L/11145–17

I/O (Combinatorial Input/Output)

TL/L/11145–18

TRI-STATE (TRI-STATE Combinatorial Output)

TL/L/11145–19

FIGURE 3

Functional Description (Continued)

In the "Small-PAL" and "Medium-PAL" modes (Table I), pins* 1 and 11 are always dedicated inputs. In the "Registered-PAL" mode, however, pin* 1 becomes the clock input controlling all OLMC registers, and pin* 11 becomes the output enable ($\overline{G}$) input controlling the TRI-STATE outputs of all registered OLMCs. Within the "Small-PAL" and "Registered-PAL" modes in Table I, the functions of pins* 12 through 19 can be selected individually from either of the two functions listed. For example, in "Registered-PAL" mode, pins* 12 through 19 can each be designated as either a registered output or a combinatorial I/O. The "Medium-PAL" mode represents a single fixed configuration used to emulate combinatorial medium PAL devices (16L8, 16H8, 16P8).

Table II lists the bipolar PAL products which the GAL16V8QS can emulate, and the specific input/output configurations used. This is just a subset, however, of all the configurations provided in Table I.

All registers in a GAL device are reset to the low state upon power-up. The active-low outputs, in turn, assume high logic levels (if enabled) regardless of the selected output polarity. This may simplify sequential circuit design and test. To ensure successful power-up reset, V_{CC} must rise monotonically until the specified operating voltage is attained. During power-up, the clock input should assume a valid, stable logic state as early as possible (within the specified time, t_{PR}) to avoid interfering with the reset operation. The clock input should also remain stable until after the power-up reset operation is completed to allow the registers to capture the proper next state on the first high-going clock transition.

It should be noted that the switching of any input not logically connected to a product term or logic function has no effect on the associated output logic state. To minimize power consumption, however, unused inputs should be connected to a stable logic level such as ground or V_{CC} (CMOS GAL inputs may be tied directly to the supply voltage without causing excessive loading conditions).

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8QS.

Clock/Input Frequency Specifications

The clock frequency (f_{CLK}) parameter listed in the Recommended Operating Conditions table specifies the maximum speed at which the GAL registers are guaranteed to operate. Clock frequency is defined differently for the two cases in which register feedback is used versus when it is not. In a data-path type application, when the logic functions fed into the registers are not dependent on register feedback from the previous cycle (i.e. based only on external inputs), the minimum required cycle period ($f_{CLK}{}^{-1}$ without feedback) is defined as the greater of the minimum clock period (t_w high + t_w low) and the minimum "data window" period (t_{SU} + t_H). This assumes optimal alignment between data inputs and the clock input. In sequential logic applications such as state machines, the minimum required cycle period ($t_{CYCLE} = f_{CLK}{}^{-1}$ with feedback) is defined as $t_{CLK} + t_{SU}$. This provides sufficient time for outputs from the registers to feed back through the logic array and set up on the inputs to the registers before the end of each cycle.

The input frequency (f_I) parameter specifies the maximum rate at which each GAL input can be toggled and still produce valid logic transitions on each combinatorial output. The f_I specification is derived as the inverse of the combinatorial propagation delay (t_{PD}).

Design Development Support

A variety of software tools and programming equipment is available to support the development of designs using GAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate a JEDEC-compatible "cell-map" (analogous to a PAL "fuse-map"). The industry-standard JEDEC format ensures that the resulting cell-map file can be down-loaded into a variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

National strongly recommends using only approved programming hardware and software for developing GAL designs. Programming using unapproved equipment generally voids all guarantees. Approved programmers incorporate specialized programming algorithms that program the array and automatically configure the architecture cells. To ensure data retention and reliability, the programming algorithm also tracks the number of programming cycles to which each GAL device has been subjected since shipment, and stores this information automatically in the device.

The special GAL programming algorithm can also program a GAL device using a standard fuse-map developed for any of the emulated PAL products. PAL fuse-maps can be created by any JEDEC-compatible PAL development software or by loading the fuse pattern from an existing programmed PAL device into the programming unit (provided the PAL device has not been secured). However, to utilize the full flexibility of the GAL architecture, true GAL development software (such as OPAL software) is recommended.

Detailed logic diagrams showing all JEDEC cell-map addresses in the GAL logic array and OLMC are provided for direct map editing and diagnostic purposes (see "Programming Details"). For a list of current software and programming support tools available for these devices, please contact your local National sales representative or distributor. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

Security Cell

A security cell is provided on all GAL16V8QS devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the array. The security cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed.

Electronic Signature

Each GAL device contains an electronic signature word consisting of 64 bits of reprogrammable memory. The electronic signature word can be programmed to contain any identification information desired by the user. Some uses include pattern identification labels, revision numbers, dates, inventory control information, etc. The data stored in the electronic signature word has no effect on the functionality of the device. The information is read out of the device using the normal program verification procedure provided by the programming equipment. The information may be accessed at any time independent of the state of the security cell. National's OPAL development software allows electronic signature data to be entered by the user and downloaded to the programming equipment.

Bulk Erase

The programming equipment automatically performs a bulk erase operation prior to each programming operation. No special erase operation need be performed by the user. Bulk erase clears the logic array, architecture cells, security cell, and electronic signature information. The GAL device is thereby reverted back to its virgin state.

Latch-Up Protection

GAL devices are designed with an on-chip charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

To insure that no undesired bias conditions occur with P+ diffusions, a Latch-Lock™ power-up circuitry has been developed. The drain of all P channel devices normally connected to the device supply are now connected to an alternate supply that powers up after the device N-wells have been biased and the substrate has reached its negative clamp value. This prevents any hazardous bias conditions from developing in the power-up sequence. After power-up is complete, the Latch-Lock circuitry becomes dormant until a full power-down has occurred; this eliminates the chance of an unwanted P channel power-down during device operation.

Manufacturer Testing

Because of EECMOS technology, GAL devices can be reprogrammed in milliseconds. This allows each device to be completely tested by the manufacturer using numerous logic array and architecture patterns prior to shipping. Every programmable cell and every logic path through every device is fully tested for programmability, functionality and performance to all AC and DC parameters. The customer can therefore expect 100% programming and functional yield and 100% compliance of all GAL products to datasheet specifications.

The testing procedure performed on all GAL devices by the manufacturer tests all aspects of device operation. Extensive testing of all programmable cells in the device include margin testing, internal verify, and program retention during high-temperature bake. All DC and AC parameters are tested at hot and cold temperatures using a variety of worst case logic and signal patterns. Functional tests include reprogramming each OLMC to all valid architectural configurations.

Register Preload

The register preload feature allows OLMC registers to be directly loaded with any desired data pattern. It also allows the present state of OLMC registers to be examined regardless of TRI-STATE control conditions. This simplifies testing of devices after programming. A device may be put into any desired register state at any point during the functional test sequence. The test sequence may then be resumed to verify proper next-state transitions. This allows complete verification of sequential logic circuits, including states that are normally impossible or difficult to reach. It may also shorten the overall test time significantly.

Register preload is not an operational mode and is not intended for board-level testing because elevated voltage levels must be applied to the device. The programming equipment normally provides the register preload capability as part of its functional test facility. Note that the testing of GAL devices after programming by the user may be considered unnecessary because all EECMOS GAL products are completely tested by the manufacturer, guaranteeing 100% post-programming functional yield.

The register preload algorithm is described for those users who wish to test programmed GAL devices using test equipment other than approved GAL programming equipment. As shown in the Register Preload Waveform in *Figure 5*, the preload sequence must not begin until the normal power-up reset operation has completed (after time t_{RESET}). The device is placed into preload mode by raising the "PRLD" input (pin* 11) to voltage V_{IES}, as specified in the Register Preload Specifications (Table III).

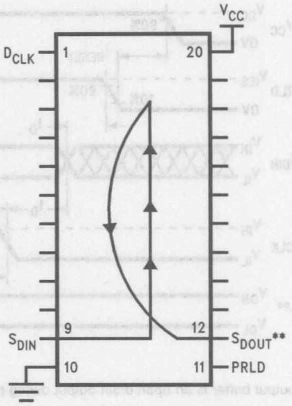
Register Preload (Continued)

To preload the OLMC registers, a series of data bits are shifted into the device on the "S_{DIN}" input (pin* 9), one bit for each OLMC in which registered output has been selected. (Non-registered OLMCs are bypassed.) The shift sequence is clocked by the rising edge of the "D_{CLK}" input (pin* 1). The data stream is shifted in through the registered OLMC with the lowest corresponding pin number, and then "upward" through all remaining registered OLMCs in pin-number ascending order. Therefore, the first data bit in the series is ultimately loaded into the registered OLMC with the highest corresponding pin number, as shown in *Figure 4*.

As the data series is shifted into the S_{DIN} input, the contents of all registers (in registered OLMCs) are shifted "upward" and out onto the "S_{DOUT}" output (pin* 12). Complete present-state information can be examined in this manner. Test fixtures can be devised to test several GAL devices in which the S_{DOUT} pin of each chip is connected to the S_{DIN} pin of the next, and all preload and present-state data can be shifted around a single serial loop.

Note that when shifting register data into S_{DIN} or out of S_{DOUT}, V_{IL}/V_{OL} = register reset (0), and V_{IH}/V_{OH} = register set (1). These 0 and 1 register states are always inverted (active-low) on the normal output pins regardless of the selected output polarity (polarity affects logic function values before register inputs).

*Applies to both 20-pin DIP and 20-lead PLCC Packages for GAL16V8QS.



TL/L/11145-20

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

FIGURE 4. Output Register Preload Pinout

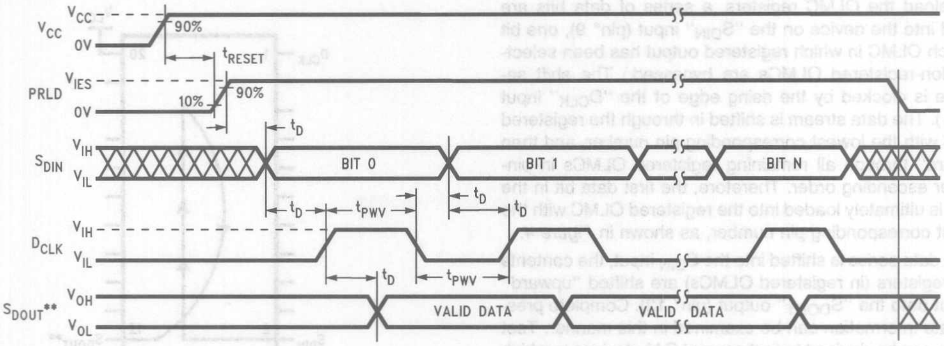
Register Preload Specifications

TABLE III

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input Voltage (High)		2.40		V_{CC}	V
V_{IL}	Input Voltage (Low)		0.00		0.50	V
V_{IES}	Registered Preload Input Voltage		14.5	15	15.5	V
V_{OH}	Output Voltage (High) (Note 1)				V_{CC}	V
V_{OL}	Output Voltage (Low) (Note 1)	$I_{OL} \leq 12$ mA	0.00		0.50	V
I_{IH}, I_{IL}	Input Current (Programming)			±1	±10	µA
I_{OH}	High Level Output Current (Note 1)	$V_{OH} \leq V_{CC}$			10	µA
t_{PWV}	Verify Pulse Width		1	5	10	µs
t_D	Pulse Sequence Delay		1	5	10	µs
t_{RESET}	Register Reset Time from Valid V_{CC}				45	µs

Note 1: The S_{DOUT} output buffer is an open drain output. This pin should be terminated to V_{CC} with a 10k resistor.

Register Preload Waveforms



TL/L/11145–21

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

FIGURE 5

Programming Details

Understanding the information in this section is not essential when using approved programming equipment and software for developing GAL designs. This is a more thorough disclosure of the GAL architecture provided for direct JEDEC cell-map editing and diagnostic purposes. This section alone, however, does not contain sufficient information to implement the GAL programming algorithm. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

As mentioned in the Functional Description, the OLMC is responsible for selecting input and/or output paths, registered vs. combinatorial outputs, active-high or low polarity, and common vs. locally-controlled TRI-STATE control. Additionally, the OLMCs select between alternate logic array input paths to maintain JEDEC cell-map compatibility with either "small-PAL" or "medium-PAL" logic arrays.

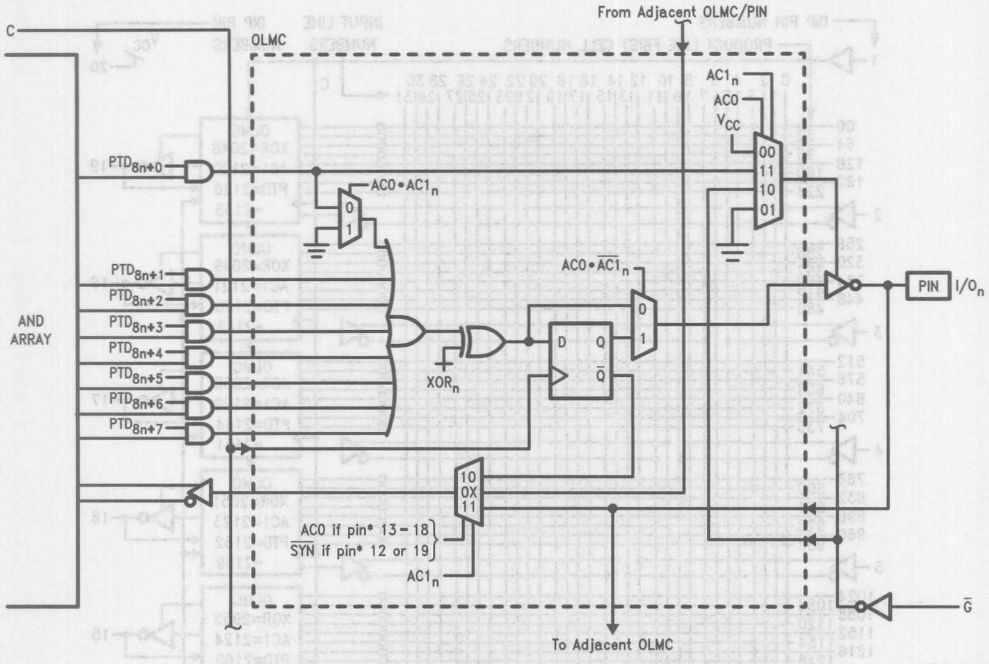
The various configurations of the OLMCs are controlled by a set of programmable "architecture" cells, separate from the logic-defining array cells. Each GAL device contains two "global" architecture cells, "SYN" and "AC0", which affect all OLMCs. Each of the device's eight OLMCs also contains two "local" cells, "AC1" and "XOR". The OLMC Logic Diagram in *Figure 6* shows how the architecture cells select the different paths through the OLMC.

The SYN bit controls whether a device will have any registered outputs (SYN = 0) or will be purely combinatorial (SYN=1). The SYN bit determines whether device pins* 1 and 11 are used as the clock and global TRI-STATE control inputs (SYN=0) or whether they are ordinary inputs (SYN=1). The AC0 bit selects between the "Small-PAL" mode and the "Medium/Registered-PAL" modes. The function of the AC1 bits depend on the state of the AC0 bit. In "Small-PAL" mode (AC0=0), the AC1 bit in each OLMC determines whether the associated device pin is an output (AC1=0) or an input (AC1=1). In "Registered-PAL" mode (AC0=1), the AC1 bit determines whether each OLMC is registered (AC1=0) or combinatorial (AC1=1). In "Medium-PAL" mode (AC0=1), the AC1 bits in all OLMCs must be set to 1 (combinatorial). All of the valid architecture bit configurations are shown in the OLMC Architecture table (Table IV), which has the same familiar format used in the OLMC Selection table (Table I).

Independent of SYN, AC0 and the AC1 bits, the XOR bit in each OLMC selects between active-low (XOR=0) or active-high (XOR=1) output polarity.

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8QS.

OLMC Logic Diagram



TL/L/11145–22

*Applies to both 20-pin DIP and 20-lead PLCC packages for GAL16V8QS.

FIGURE 6

OLMC Architecture Programming

TABLE IV

	"Small-PAL" Mode			"Registered-PAL" Mode			"Medium-PAL" Mode	
	Function		JEDEC Input Line #s (Note 1)	Function		JEDEC Input Line #s (Note 1)	Function	JEDEC Input Lines #s (Note 1)
Pin 1	INPUT	INPUT	2,3	CLOCK	CLOCK		INPUT	2,3
*** Pin 19	I/O	INPUT	6,7	REGISTER	I/O	2,3	TRI-STATE**	
*** Pin 18	I/O	INPUT	10,11	REGISTER	I/O	6,7	I/O	6,7
*** Pin 17	I/O	INPUT	14,15	REGISTER	I/O	10,11	I/O	10,11
*** Pin 16	OUTPUT*	NC		REGISTER	I/O	14,15	I/O	14,15
*** Pin 15	OUTPUT*	NC		REGISTER	I/O	18,19	I/O	18,19
*** Pin 14	I/O	INPUT	18,19	REGISTER	I/O	22,23	I/O	22,23
*** Pin 13	I/O	INPUT	22,23	REGISTER	I/O	26,27	I/O	26,27
*** Pin 12	I/O	INPUT	26,27	REGISTER	I/O	30,13	TRI-STATE**	
Pin 11	INPUT	INPUT	30,31	$\overline{G}$	$\overline{G}$		INPUT	30,31
	$AC1_n = 0$	$AC1_n = 1$		$AC1_n = 0$	$AC1_n = 1$		$AC1_n = 1$	
	SYN = 1, AC0 = 0			SYN = 0, AC0 = 1			SYN = 1, AC0 = 1	
	All outputs are combinatorial and always active.			At least one output is registered.			All I/O pins are combinatorial.	

Note: Pin numbers above apply to both 20-pin DIP and 20-lead PLCC packages for GAL16V8QS.

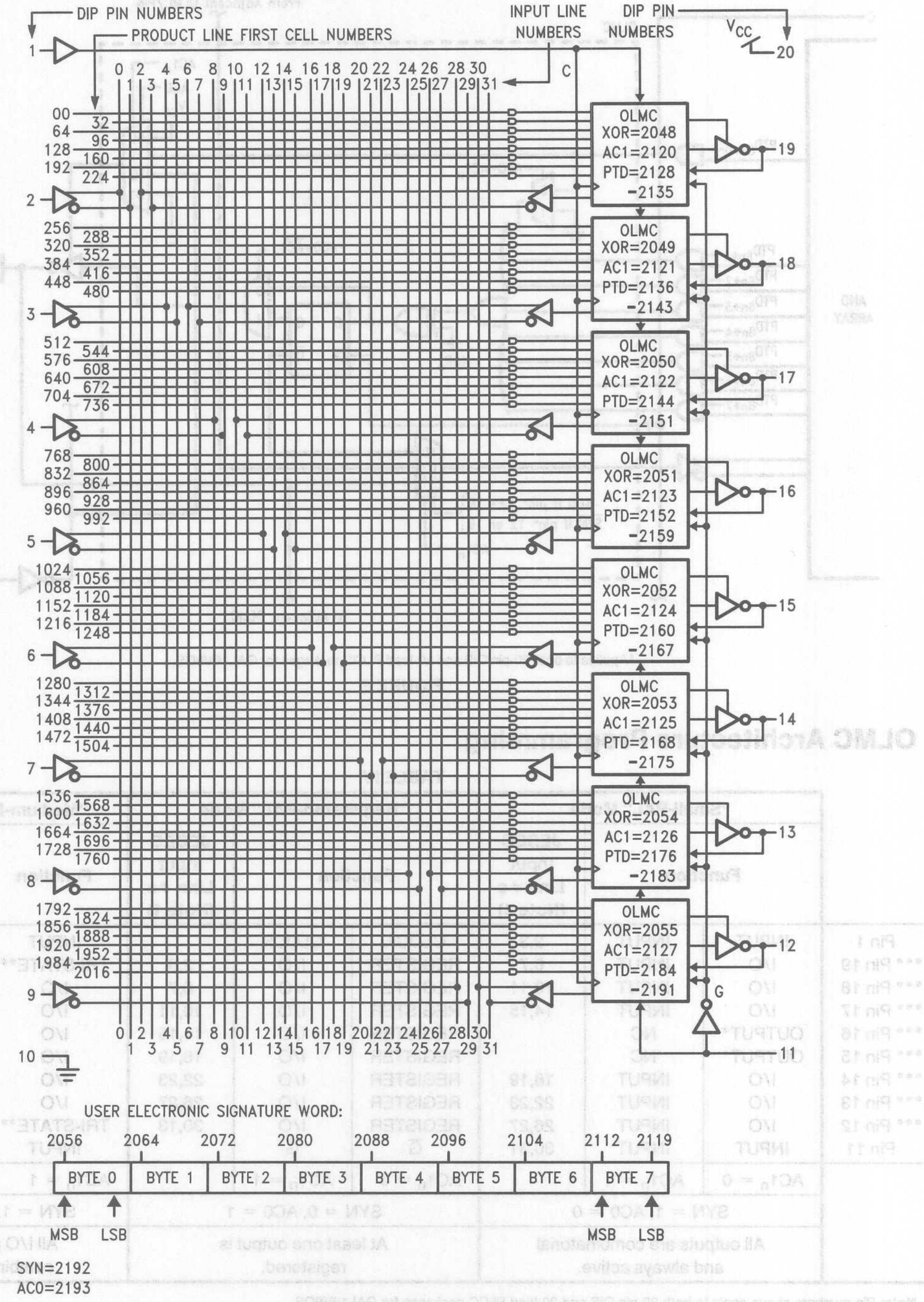
Note 1: All even and odd numbered JEDEC input line numbers correspond to true and complement array inputs, respectively.

*Active combinatorial output.

**TRI-STATE combinatorial output.

*** $AC1_n$ applies to these I/O pins only.

GAL16V8QS Logic Diagram



TL/L/11145–23

JEDEC Logic Array Cell Number = Product Line First Cell Number + Input Line Number

FIGURE 7

National Semiconductor

GAL20V8QS 24-Pin Generic Array Logic Family

General Description

The EECMOS GAL® QS™ devices combine a high performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The GALQS family utilizes NSC Quiet Series technology to guarantee quiet output switching and improve dynamic threshold performance. GAL Quiet Series™ features GTO™ output control in addition to a split ground bus for superior performance.

The 24-pin GAL20V8QS features 8 programmable Output Logic Macrocells (OLMCs) allowing each TRI-STATE® output to be configured by the user. Additionally, the GAL20V8QS is capable of emulating, in a functional/fuse map/parametric compatible device, the most popular 24-pin PAL® device architectures.

Programming is accomplished using readily available hardware and software tools. NSC guarantees a minimum 100 erase/write cycles.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, NSC guarantees 100% field programmability of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

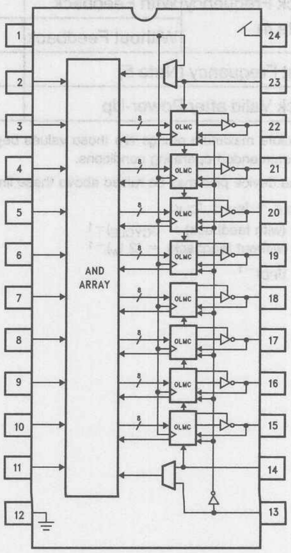
Features

- High performance EECMOS technology
  - — 15 ns maximum propagation delay
  - — f_{CLK} = 62.5 MHz
  - — 10 ns maximum from clock input to data output
  - — TTL compatible 24 mA outputs
  - — UltraMOS® III advanced CMOS technology
- Guaranteed simultaneous switching noise level and dynamic threshold performance
- Reduced Groundbounce and Undershoot
- Reduced power
  - — Low power = 90 mA I_{CC} max
  - — Quarter power = 55 mA I_{CC} max
- Electrically erasable cell technology
  - — Reconfigurable logic
  - — Reprogrammable cells
  - — 100% tested/guaranteed 100% yields
  - — High speed electrical erasure (<50 ms)
  - — 20 year data retention
- Eight output logic macrocells
  - — Maximum flexibility for complex logic designs
  - — Programmable output polarity
  - — Also emulates 24-pin PAL devices with full function/fuse map/parametric compatibility
- Preload and power-up reset of all registers
  - — 100% functional testability
- Fully supported by National OPAL™ and OPALjr development software
- Security cell prevents copying logic
- Electronic signature for identification

PAL Replacement by Device Type

"Small PAL" Mode				"Registered PAL" Mode			"Medium PAL" Mode
14L8 14H8 14P8	16L6 16H6 16P6	18L4 18H4 18P4	20L2 20H2 20P2	20R8 20RP8	20R6 20RP6	20R4 20RP4	20L8 20H8 20P8

Block Diagram—GAL20V8QS



TL/L/11144-1

GAL20V8QS (-15L, -15Q, -20L, -25L, -25Q) Commercial

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Units
		Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	V
T_A	Operating Free-Air Temperature	0	25	75	°C
T_C	Operating Case Temperature				°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL20V8QS-15L		GAL20V8QS-15Q		GAL20V8QS-20L		GAL20V8QS-25L/25Q		Units
			COM		COM		COM		COM		
			Min	Max	Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		12		12		15		15		ns
t_H	Hold Time (Input after Clock)		0		0		0		0		ns
t_w	Clock Pulse Width (High/Low)		8		8		12		12		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		22		22		27		27		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		45.5		45.5		37		37	MHz
		Without Feedback		62.5		62.5		41.66		41.66	
f_I	Input Frequency (Note 5)			66.6		66.6		50		40	
t_{PR}	Clock Valid after Power-Up			100		100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) $= (t_{CYCLE})^{-1}$
f_{CLK} (without feedback) $= (2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL20V8QS (-15L, -15Q, -20L, -25L, -25Q) Commercial (Continued)

Quiet Electrical Characteristics (V_{CC} = 5.0V, Temp = 25°C)

Symbol	Parameter	Conditions	Typ	Max	Units
V_{OLP}	Quiet Output Maximum Dynamic V_{OL}	(Note 6)	1.1	1.5	V
V_{OLV}	Quiet Output Minimum Dynamic V_{OL}	(Note 6)	−0.6	−1.2	V
V_{IHD}	Maximum High Level Dynamic Input Voltage	(Note 7)	1.9	2.2	V
V_{ILD}	Maximum Low Level Dynamic Input Voltage	(Note 7)	1.2	0.8	V

Note 6: 7 output switching from high to low, one output at low. The width of the bounce at 50% amplitude is ≤3 ns.

Note 7: 7 output switching. Input__under__test switching:
3V to threshold (V_{ILD}), 0V to threshold (V_{IHD})
f = 1 MHz

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		V_{CC} + 1	V
V_{IL}	Low Level Input Voltage				−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA	COM	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 24 mA	COM			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)					10	µA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	µA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	µA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	µA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	µA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	-15L, -20L, -25L	COM			90	mA
			-15Q, -25Q	COM			55	
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V				5	8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V				10	10	pF

*One output at a time for a maximum duration of one second.

GAL20V8QS (-15L, -15Q, -20L, -25L, -25Q) Commercial (Continued)

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL20V8QS-15L/-15Q		GAL20V8QS-20L		GAL20V8QS-25L/-25Q		Units
			COM		COM		COM		
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, $C_L = 50$ pF		15		20		25	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, $C_L = 50$ pF		10		12		12	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High; S1 Open, $C_L = 50$ pF Active Low; S1 Closed, $C_L = 50$ pF		15		18		20	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}; S1 Open, $C_L = 5$ pF From V_{OL}; S1 Closed, $C_L = 5$ pF		15		18		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High; S1 Open, $C_L = 50$ pF Active Low; S1 Closed, $C_L = 50$ pF		15		20		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}; S1 Open, $C_L = 5$ pF From V_{OL}; S1 Closed, $C_L = 5$ pF		15		20		25	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, $C_L = 50$ pF		45		45		45	µs

GAL20V8QS (-20L, -25L, -25Q) Industrial

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C

Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028 Rev. C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Industrial			Units
		Min	Typ	Max	
V_{CC}	Supply Voltage	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	−40	25	85	°C
T_C	Operating Case Temperature				°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL20V8QS-20L		GAL20V8QS-25L		GAL20V8QS-25Q		Units
			IND		IND		IND		
			Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		15		20		20		ns
t_H	Hold Time (Input after Clock)		0		0		0		ns
t_w	Clock Pulse Width (High/Low)		12		15		15		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		30		35		35		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		33.3		28.5		28.5	MHz
		Without Feedback		41.66		33.3		33.3	
f_I	Input Frequency (Note 5)			50		40		40	
t_{PR}	Clock Valid after Power-Up			100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) = $(t_{CYCLE})^{-1}$
f_{CLK} (without feedback) = $(2\ t_w)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

GAL20V8QS (-20L, -25L, -25Q) Industrial (Continued)

Quiet Electrical Characteristics (V_{CC} = 5.0V, Temp = 25°C)

Symbol	Parameter	Conditions	Typ	Max	Units
V_{OLP}	Quiet Output Maximum Dynamic V_{OL}	(Note 6)	1.1	1.5	V
V_{OLV}	Quiet Output Minimum Dynamic V_{OL}	(Note 6)	−0.6	−1.2	V
V_{IHD}	Maximum High Level Dynamic Input Voltage	(Note 7)	1.9	2.2	V
V_{ILD}	Maximum Low Level Dynamic Input Voltage	(Note 7)	1.2	0.8	V

Note 6: 7 output switching from high to low, one output at low. The width of the bounce at 50% amplitude is ≤3 ns.

Note 7: 7 output switching. Input__under__test switching:
3V to threshold (V_{ILD}), 0V to threshold (V_{IHD})
f = 1 MHz

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		V_{CC} + 1	V
V_{IL}	Low Level Input Voltage				−0.5		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA	IND	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 24 mA	IND			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)					10	µA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	µA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC}(Max)					10	µA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC}(Max)					10	µA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	µA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	-20L/-25L	IND			90	mA
			-25Q	IND			55	
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V				5	8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V				8	10	pF

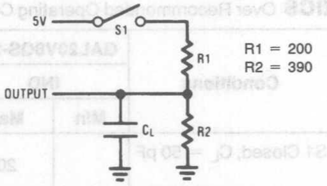
*One output at a time for a maximum duration of one second.

GAL20V8QS (-20L, -25L, -25Q) Industrial (Continued)

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL20V8QS-20L		GAL20V8QS-25L		GAL20V8QS-25Q		Units
			IND		IND		IND		
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, $C_L = 50$ pF		20		25		25	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, $C_L = 50$ pF		15		15		15	ns
t_{PZXG}	$\overline{G}$ ↓ to Registered Output Enabled	Active High; S1 Open, $C_L = 50$ pF Active Low; S1 Closed, $C_L = 50$ pF		18		20		20	ns
t_{PXZG}	$\overline{G}$ ↑ to Registered Output Disabled	From V_{OH}; S1 Open, $C_L = 5$ pF From V_{OL}; S1 Closed, $C_L = 5$ pF		18		20		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High; S1 Open, $C_L = 50$ pF Active Low; S1 Closed, $C_L = 50$ pF		20		25		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}; S1 Open, $C_L = 5$ pF From V_{OL}; S1 Closed, $C_L = 5$ pF		20		25		25	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, $C_L = 50$ pF		45		45		45	ns

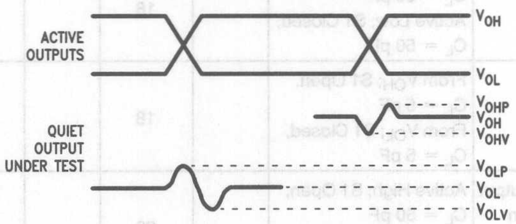
AC Test Load



TL/L/11144-2

Test Waveforms

Quiet Output Noise Voltage Waveforms



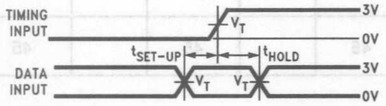
TL/L/11144-24

Note A. V_{OHV} and V_{OLP} are measured with respect to ground reference.

Note B. Input pulses have the following characteristics: f = 1 MHz, t_r = 3 ns, t_f = 3 ns, skew < 150 ps.

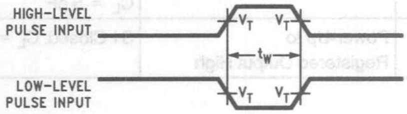
Note C. Test load for Quiet output: C_L = 50 pF, R_L = 500Ω

Setup and Hold



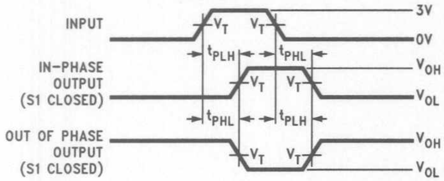
TL/L/11144-3

Pulse Width



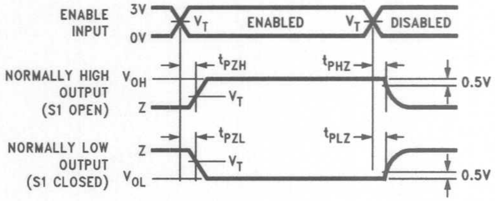
TL/L/11144-4

Propagation Delay



TL/L/11144-5

Enable and Disable



TL/L/11144-6

Notes:

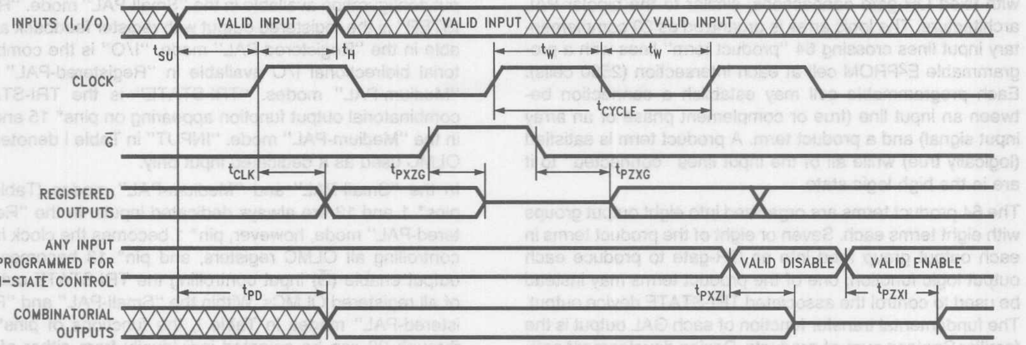
C_L includes probe and jig capacitance.

V_T = 1.5V.

Test inputs have rise and fall times of 5 ns between 0.3V and 2.7V.

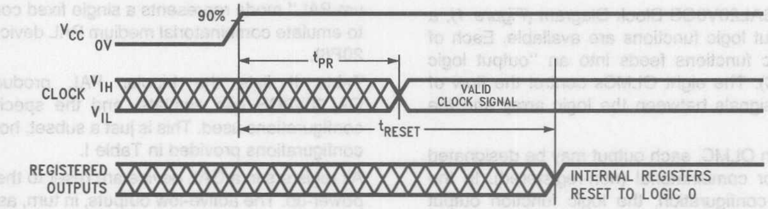
In the example above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Switching Waveforms



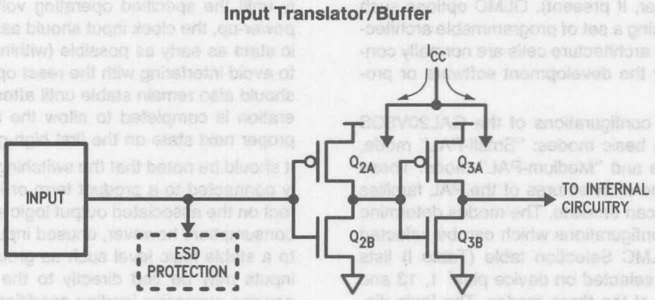
TL/L/11144-7

Power-Up Reset Waveforms



TL/L/11144-8

Input Schematic



TL/L/11144-9

Functional Description

The GAL logic array consists of a programmable AND array with fixed OR-gate connections, similar to the bipolar PAL architecture. The logic array is organized as 20 complementary input lines crossing 64 "product term" lines with a programmable E^2PROM cell at each intersection (2560 cells). Each programmable cell may establish a connection between an input line (true or complement phase of an array input signal) and a product term. A product term is satisfied (logically true) while all of the input lines "connected" to it are in the high logic state.

The 64 product terms are organized into eight output groups with eight terms each. Seven or eight of the product terms in each output group feed into an OR-gate to produce each output logic function; one of the product terms may instead be used to control the associated TRI-STATE device output. The fundamental transfer function of each GAL output is the familiar Boolean sum-of-products. Design development software is available which accepts Boolean equations and converts them automatically into GAL programming patterns.

As shown in the GAL20V8QS Block Diagram *(Figure 1)*, a total of eight output logic functions are available. Each of the AND/OR logic functions feeds into an "output logic macrocell" (OLMC). The eight OLMCs control the flow of input and output signals between the logic array and the device's I/O pins.

Under control of an OLMC, each output may be designated either registered or combinatorial (non-registered). In the registered output configuration, the logic function output passes through a D-type flip-flop triggered by the rising edge of the clock input. Additionally, the logic function's output polarity may be designated active-low or active-high (adjusted before the register, if present). OLMC options such as these are selected using a set of programmable architecture control cells. These architecture cells are normally configured automatically by the development software or programming hardware.

All of the possible I/O configurations of the GAL20V8QS are classified into three basic modes: "Small-PAL" mode, "Registered-PAL" mode and "Medium-PAL" mode. These modes correspond to the architectures of the PAL families which the GAL20V8QS can emulate. The modes determine the mixture of OLMC configurations which can be selected for the device. The OLMC Selection table (Table I) lists which functions can be selected on device pins* 1, 13 and 15 through 22 for each of the three modes. The logic diagrams in *Figure 3* illustrate these OLMC functions.

"OUTPUT" represents the always-active combinatorial output configuration available in the "Small-PAL" mode. "REGISTER" is the registered output with register feedback available in the "Registered-PAL" mode. "I/O" is the combinatorial bidirectional I/O available in "Registered-PAL" and "Medium-PAL" modes. "TRI-STATE" is the TRI-STATE combinatorial output function appearing on pins* 15 and 22 in the "Medium-PAL" mode. "INPUT" in Table I denotes an OLMC used as a dedicated input only.

In the "Small-PAL" and "Medium-PAL" modes (Table I), pins* 1 and 13 are always dedicated inputs. In the "Registered-PAL" mode, however, pin* 1 becomes the clock input controlling all OLMC registers, and pin* 13 becomes the output enable ($\overline{G}$) input controlling the TRI-STATE outputs of all registered OLMCs. Within the "Small-PAL" and "Registered-PAL" modes in Table I, the functions of pins* 15 through 22 can be selected individually from either of the two functions listed. For example, in "Registered-PAL" mode, pins* 15 through 22 can each be designated as either a registered output or a combinatorial I/O. The "Medium-PAL" mode represents a single fixed configuration used to emulate combinatorial medium PAL devices (20L8, 20H8, 20P8).

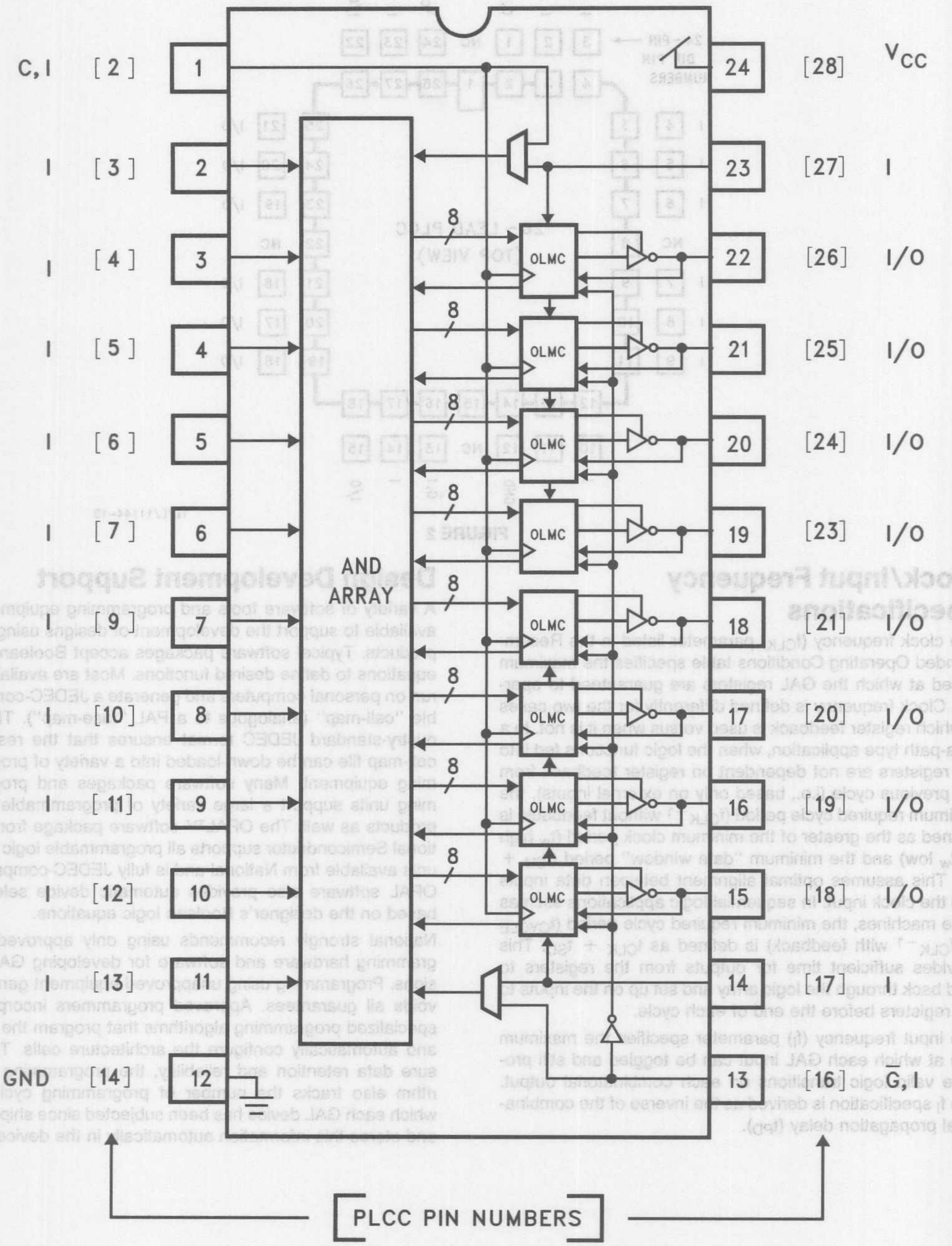
Table II lists the bipolar PAL products which the GAL20V8QS can emulate, and the specific input/output configurations used. This is just a subset, however, of all the configurations provided in Table I.

All registers in a GAL device are reset to the low state upon power-up. The active-low outputs, in turn, assume high logic levels (if enabled) regardless of the selected output polarity. This may simplify sequential circuit design and test. To ensure successful power-up reset, V_{CC} must rise monotonically until the specified operating voltage is attained. During power-up, the clock input should assume a valid, stable logic state as early as possible (within the specified time, t_{PR}) to avoid interfering with the reset operation. The clock input should also remain stable until after the power-up reset operation is completed to allow the registers to capture the proper next state on the first high-going clock transition.

It should be noted that the switching of any input not logically connected to a product term or logic function has no effect on the associated output logic state. To minimize power consumption, however, unused inputs should be connected to a stable logic level such as ground or V_{CC} (CMOS GAL inputs may be tied directly to the supply voltage without causing excessive loading conditions).

*Applies to 24-pin DIP packages for GAL20V8QS; refer to the 28-lead PLCC Connection Diagram for conversion.

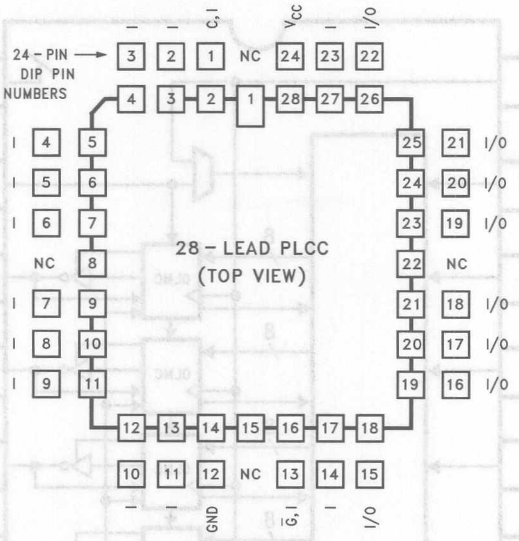
GAL20V8QS Block Diagram—DIP Connections



TL/L/11144–12

FIGURE 1

28-Lead PLCC Connection Diagram



TL/L/11144-13

FIGURE 2

Clock/Input Frequency Specifications

The clock frequency (f_{CLK}) parameter listed in the Recommended Operating Conditions table specifies the maximum speed at which the GAL registers are guaranteed to operate. Clock frequency is defined differently for the two cases in which register feedback is used versus when it is not. In a data-path type application, when the logic functions fed into the registers are not dependent on register feedback from the previous cycle (i.e., based only on external inputs), the minimum required cycle period (f_{CLK}^{-1} without feedback) is defined as the greater of the minimum clock period (t_W high + t_W low) and the minimum "data window" period (t_{SU} + t_H). This assumes optimal alignment between data inputs and the clock input. In sequential logic applications such as state machines, the minimum required cycle period (t_{CYCLE} = f_{CLK}^{-1} with feedback) is defined as t_{CLK} + t_{SU}. This provides sufficient time for outputs from the registers to feed back through the logic array and set up on the inputs to the registers before the end of each cycle.

The input frequency (f_I) parameter specifies the maximum rate at which each GAL input can be toggled and still produce valid logic transitions on each combinatorial output. The f_I specification is derived as the inverse of the combinatorial propagation delay (t_{PD}).

Design Development Support

A variety of software tools and programming equipment is available to support the development of designs using GAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate a JEDEC-compatible "cell-map" (analogous to a PAL "fuse-map"). The industry-standard JEDEC format ensures that the resulting cell-map file can be down-loaded into a variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL™ software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

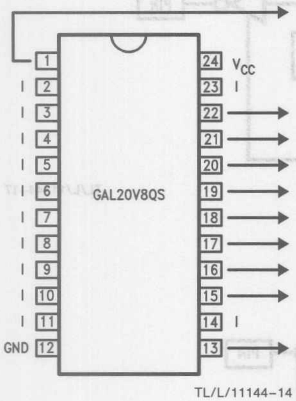
National strongly recommends using only approved programming hardware and software for developing GAL designs. Programming using unapproved equipment generally voids all guarantees. Approved programmers incorporate specialized programming algorithms that program the array and automatically configure the architecture cells. To ensure data retention and reliability, the programming algorithm also tracks the number of programming cycles to which each GAL device has been subjected since shipment, and stores this information automatically in the device.

Design Development Support (Continued)

The special GAL programming algorithm can also program a GAL device using a standard fuse-map developed for any of the emulated PAL products. PAL fuse-maps can be created by any JEDEC-compatible PAL development software or by loading the fuse pattern from an existing programmed PAL device into the programming unit (provided the PAL device has not been secured). However, to utilize the full flexibility of the GAL architecture, true GAL development software (such as OPAL software) is recommended.

Detailed logic diagrams showing all JEDEC cell-map addresses in the GAL logic array and OLMC are provided for direct map editing and diagnostic purposes (see "Programming Details"). For a list of current software and programming support tools available for these devices, please contact your local National sales representative or distributor. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

OLMC Selection Table



TL/L/11144–14

TABLE I

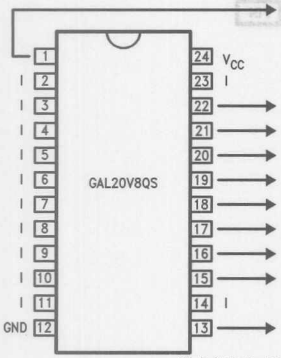
"Small-PAL" Mode	"Registered-PAL" Mode	"Medium-PAL" Mode
INPUT	CLOCK	INPUT
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	I/O
INPUT or OUTPUT*	REGISTER or I/O	TRI-STATE**
INPUT	OUTPUT ENABLE ($\overline{G}$)	INPUT

*Active combinatorial output

**TRI-STATE combinatorial output

Note: Pin numbers above apply to 24-pin DIP packages; refer to the 28-lead PCC Connection Diagram for conversion.

PAL Replacement Configurations



TL/L/11144–15

TABLE II

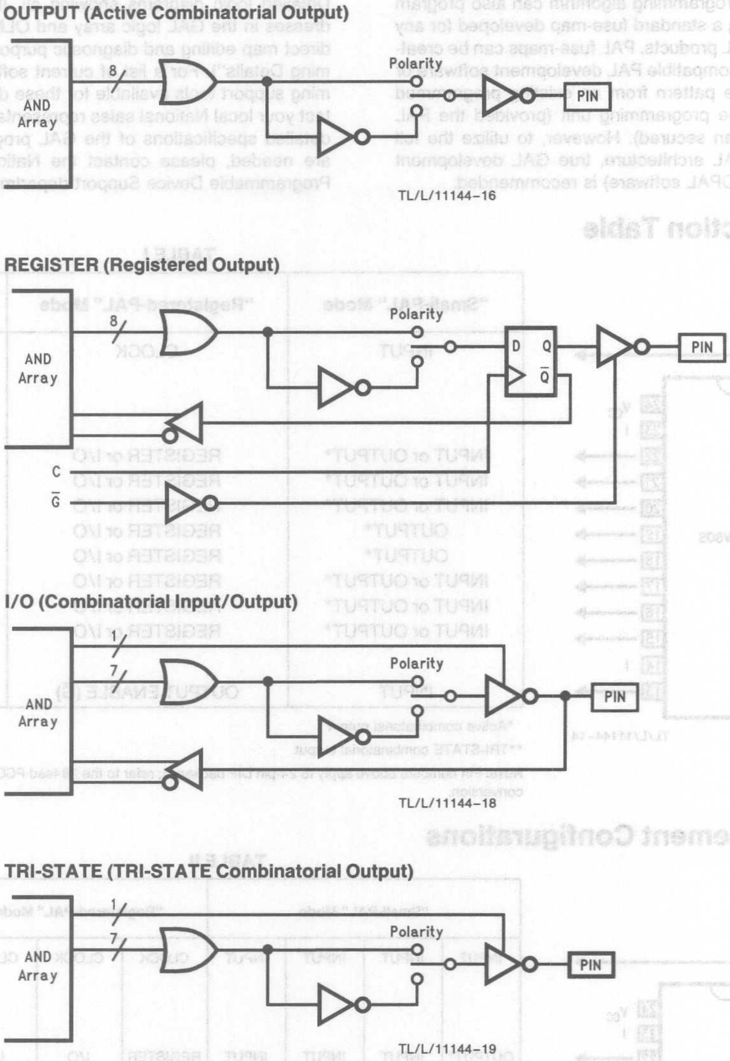
	"Small-PAL" Mode				"Registered-PAL" Mode			"Medium-PAL" Mode
	INPUT	INPUT	INPUT	INPUT	CLOCK	CLOCK	CLOCK	INPUT
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	OUTPUT*	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	OUTPUT*	INPUT	REGISTER	REGISTER	REGISTER	I/O
	OUTPUT*	OUTPUT*	INPUT	INPUT	REGISTER	REGISTER	I/O	I/O
	OUTPUT*	INPUT	INPUT	INPUT	REGISTER	I/O	I/O	TRI-STATE**
	INPUT	INPUT	INPUT	INPUT	$\overline{G}$	$\overline{G}$	$\overline{G}$	INPUT
	14L8	16L6	18L4	20L2	20R8	20R6	20R4	20L8
Emulated	14H8	16H6	18H4	20H2	20RP8	20RP6	20RP4	20H8
PAL Products	14P8	16P6	18P4	20P2				20P8

* Active combinatorial output.

**TRI-STATE combinatorial output.

Note: Pin numbers above apply to 24-pin DIP packages; refer to the 28-pin PCC Connection Diagram for conversion.

OLMC Configurations



OUTPUT (Active Combinatorial Output)

REGISTER (Registered Output)

I/O (Combinatorial Input/Output)

TRI-STATE (TRI-STATE Combinatorial Output)

FIGURE 3

Security Cell

A security cell is provided on all GAL20V8QS devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the array. The security cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed.

Electronic Signature

Each GAL device contains an electronic signature word consisting of 64 bits of reprogrammable memory. The electronic signature word can be programmed to contain any identification information desired by the user. Some uses include pattern identification labels, revision numbers, dates, inventory control information, etc. The data stored in the electronic signature word has no effect on the functionality of the device. The information is read out of the device using the normal program verification procedure provided by the programming equipment. The information may be accessed at any time independent of the state of the security cell. National's OPAL development software allows electronic signature data to be entered by the user and downloaded to the programming equipment.

Bulk Erase

The programming equipment automatically performs a bulk erase operation prior to each programming operation. No special erase operation need be performed by the user. Bulk erase clears the logic array, architecture cells, security cell, and electronic signature information. The GAL device is thereby reverted back to its virgin state.

Latch-Up Protection

GAL devices are designed with an on-chip charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

To insure that no undesired bias conditions occur with P+ diffusions, a Latch-Lock™ power-up circuitry has been developed. The drain of all P channel devices normally connected to the device supply are now connected to an alternate supply that powers up after the device N-wells have been biased and the substrate has reached its negative clamp value. This prevents any hazardous bias conditions from developing in the power-up sequence. After power-up is complete, the Latch-Lock circuitry becomes dormant until a full power-down has occurred; this eliminates the chance of an unwanted P channel power-down during device operation.

Manufacturer Testing

Because of EECMOS technology, GAL devices can be reprogrammed in milliseconds. This allows each device to be completely tested by the manufacturer using numerous logic array and architecture patterns prior to shipping. Every programmable cell and every logic path through every device is fully tested for programmability, functionality and performance to all AC and DC parameters. The customer can therefore expect 100% programming and functional yield and 100% compliance of all GAL products to datasheet specifications.

The testing procedure performed on all GAL devices by the manufacturer tests all aspects of device operation. Extensive testing of all programmable cells in the device include margin testing, internal verify, and program retention during high-temperature bake. All DC and AC parameters are tested at hot and cold temperatures using a variety of worst-case logic and signal patterns. Functional tests include reprogramming each OLMC to all valid architectural configurations.

Register Preload

The register preload feature allows OLMC registers to be directly loaded with any desired data pattern. It also allows the present state of OLMC registers to be examined regardless of TRI-STATE control conditions. This simplifies testing of devices after programming. A device may be put into any desired register state at any point during the functional test sequence. The test sequence may then be resumed to verify proper next-state transitions. This allows complete verification of sequential logic circuits, including states that are normally impossible or difficult to reach. It may also shorten the overall test time significantly.

Register preload is not an operational mode and is not intended for board-level testing because elevated voltage levels must be applied to the device. The programming equipment normally provides the register preload capability as part of its functional test facility. Note that the testing of GAL devices after programming by the user may be considered unnecessary because all EECMOS GAL products are completely tested by the manufacturer, guaranteeing 100% post-programming functional yield.

The register preload algorithm is described for those users who wish to test programmed GAL devices using test equipment other than approved GAL programming equipment. As shown in the Register Preload Waveform in *Figure 5,* the preload sequence must not begin until the normal power-up reset operation has completed (after time t_{RESET}). The device is placed into preload mode by raising the "PRLD" input (pin* 13) to voltage V_{IES}, as specified in the Register Preload Specifications (Table III).

To preload the OLMC registers, a series of data bits are shifted into the device on the "S_{DIN}" input (pin* 11), one bit for each OLMC in which registered output has been selected. (Non-registered OLMCs are bypassed.) The shift sequence is clocked by the rising edge of the "D_{CLK}" input (pin* 1). The data stream is shifted in through the registered OLMC with the lowest corresponding pin number, and then "upward" through all remaining registered OLMCs in pin-number ascending order. Therefore, the first data bit in the series is ultimately loaded into the registered OLMC with the highest corresponding pin number, as shown in *Figure 4.*

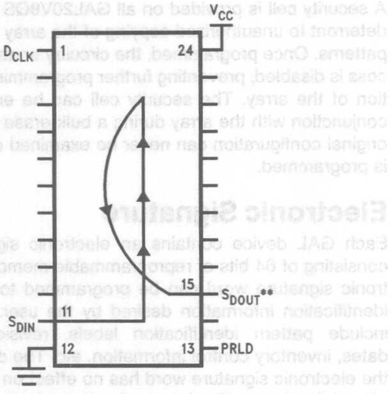
*Applies to 24-pin DIP packages for GAL20V8QS; refer to the 28-lead PLCC Connection Diagram for conversion.

Register Preload (Continued)

As the data series is shifted into the S_{DIN} input, the contents of all registers (in registered OLMCs) are shifted "upward" and out onto the "S_{DOUT}" output (pin* 15). Complete present-state information can be examined in this manner. Test fixtures can be devised to test several GAL devices in which the S_{DOUT} pin of each chip is connected to the S_{DIN} pin of the next, and all preload and present-state data can be shifted around a single serial loop.

Note that when shifting register data into S_{DIN} or out of S_{DOUT}, V_{IL}/V_{OL} = register reset (0), and V_{IH}/V_{OH} = register set (1). These 0 and 1 register states are always inverted (active-low) on the normal output pins regardless of the selected output polarity (polarity affects logic function values before register inputs).

*Applies to 24-pin DIP packages for GAL20V8QS; refer to the 28-lead PLCC Connection Diagram for conversion.



TL/L/11144-20

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

FIGURE 4. Output Register Preload Pinout

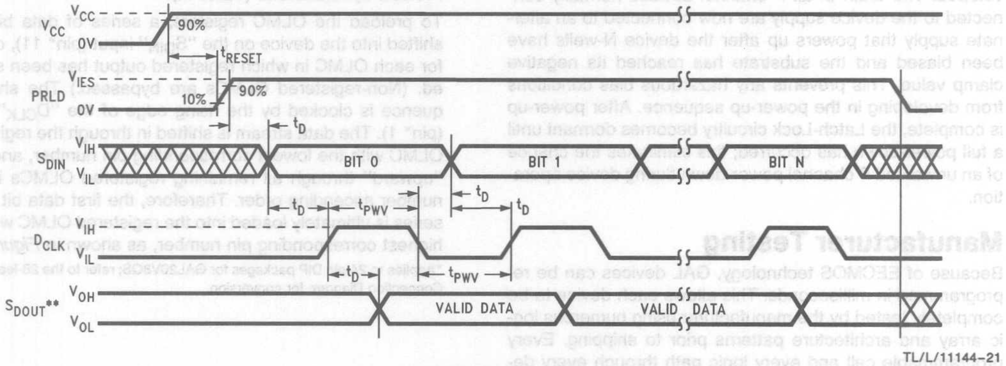
Register Preload Specifications

TABLE III

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input Voltage (High)		2.40		V_{CC}	V
V_{IL}	Input Voltage (Low)		0.00		0.50	V
V_{IES}	Register Preload Input Voltage		14.5	15	15.5	V
V_{OH}	Output Voltage (High) (Note 1)				V_{CC}	V
V_{OL}	Output Voltage (Low) (Note 1)	$I_{OL} \leq 12$ mA	0.00		0.50	V
I_{IH}, I_{IL}	Input Current (Programming)			±1	±10	μA
I_{OH}	High Level Output Current (Note 1)	$V_{OH} \leq V_{CC}$			10	μA
t_{PWV}	Verify Pulse Width			5	10	μs
t_D	Pulse Sequence Delay		1	5	10	μs
t_{RESET}	Register Reset Time from Valid V_{CC}				45	μs

Note 1: The S_{DOUT} output buffer is an open drain output. This pin should be terminated to V_{CC} with a 10k resistor.

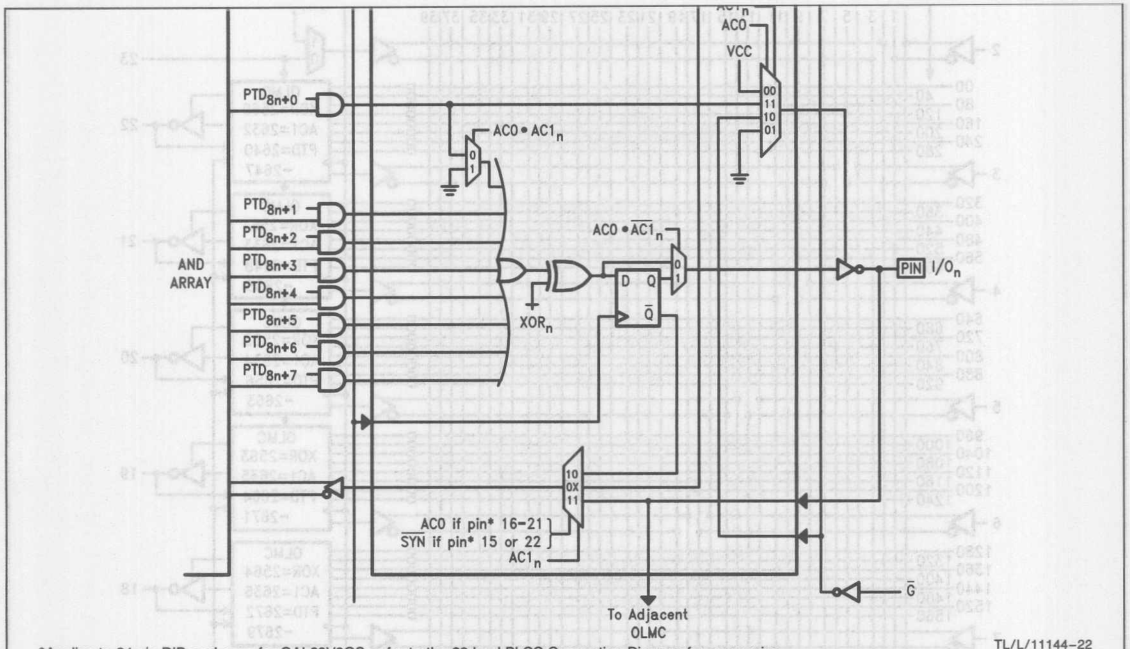
Register Preload Waveforms



TL/L/11144-21

FIGURE 5

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.



*Applies to 24-pin DIP packages for GAL20V8QS; refer to the 28-lead PLCC Connection Diagram for conversion.

TL/L/11144–22

FIGURE 6

OLMC Architecture Programming

TABLE IV

	"Small-PAL" Mode			"Registered-PAL" Mode			"Medium-PAL" Mode	
	Function		JEDEC Input Line #s (Note 1)	Function		JEDEC Input Line #s (Note 1)	Function	JEDEC Input Line #s (Note 1)
Pin 1	INPUT	INPUT	2, 3	CLOCK	CLOCK		INPUT	2, 3
Pin 23	INPUT	INPUT	6, 7	INPUT	INPUT	2, 3	INPUT	6, 7
***Pin 22	I/O	INPUT	10, 11	REGISTER	I/O	6, 7	TRI-STATE**	
***Pin 21	I/O	INPUT	14, 15	REGISTER	I/O	10, 11	I/O	10, 11
***Pin 20	I/O	INPUT	18, 19	REGISTER	I/O	14, 15	I/O	14, 15
***Pin 19	OUTPUT*	NC		REGISTER	I/O	18, 19	I/O	18, 19
***Pin 18	OUTPUT*	NC		REGISTER	I/O	22, 23	I/O	22, 23
***Pin 17	I/O	INPUT	22, 23	REGISTER	I/O	26, 27	I/O	26, 27
***Pin 16	I/O	INPUT	26, 27	REGISTER	I/O	30, 31	I/O	30, 31
***Pin 15	I/O	INPUT	30, 31	REGISTER	I/O	34, 35	TRI-STATE**	
Pin 14	INPUT	INPUT	34, 35	INPUT	INPUT	38, 39	INPUT	34, 35
Pin 13	INPUT	INPUT	38, 39	$\overline{G}$	$\overline{G}$		INPUT	38, 39
	$AC1_n = 0$	$AC1_n = 1$		$AC1_n = 0$	$AC1_n = 1$		$AC1_n = 1$	
	SYN = 1, AC0 = 0			SYN = 0, AC0 = 1			SYN = 1, AC0 = 1	
	All outputs are combinatorial and always active.			At least one output is registered.			All I/O pins are combinatorial.	

Note: Pin numbers above apply to 24-pin DIP packages; refer to the 28-lead PLCC Connection Diagram for conversion.

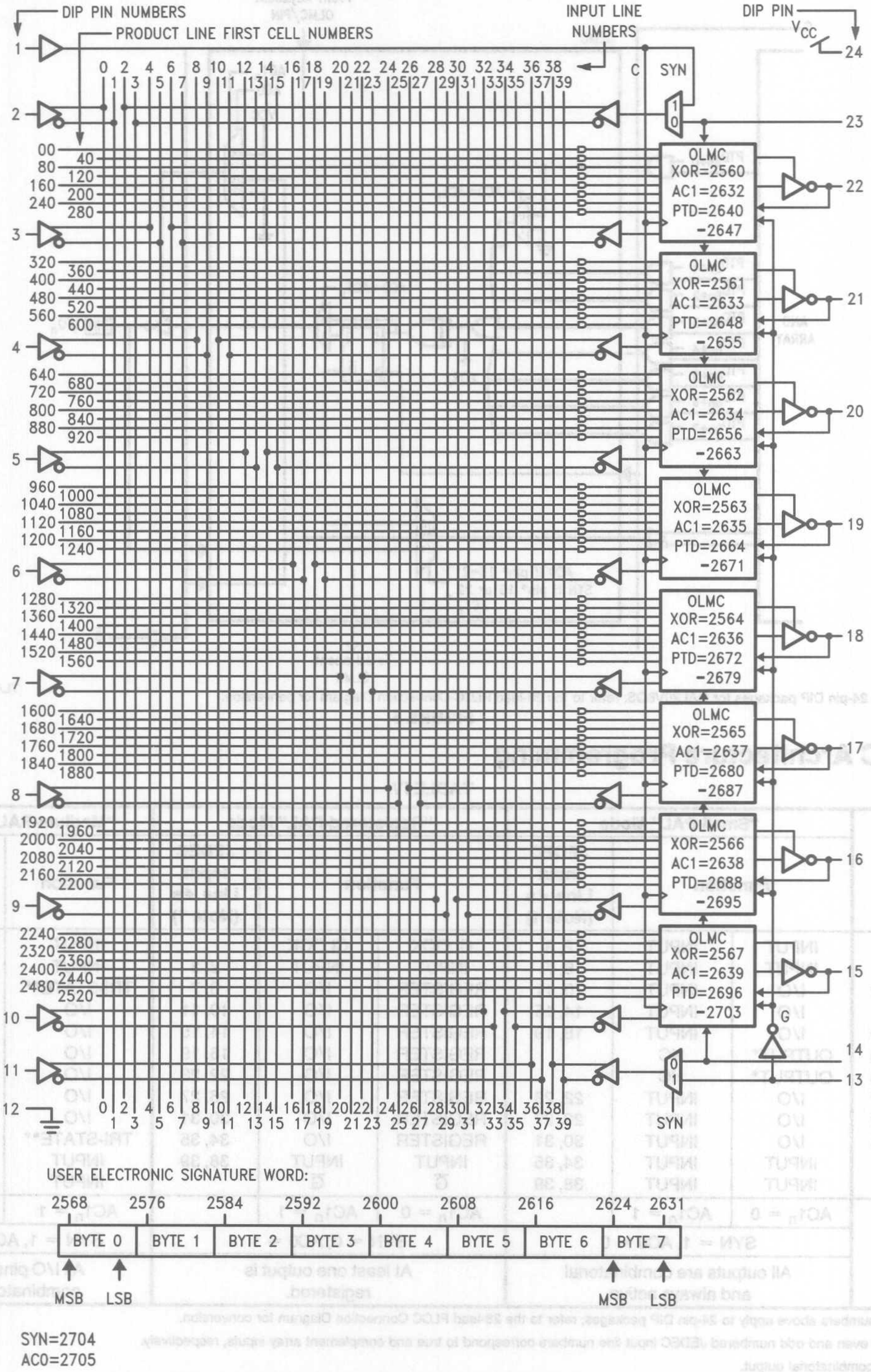
Note 1: All even and odd numbered JEDEC input line numbers correspond to true and complement array inputs, respectively.

*Active combinatorial output.

**TRI-STATE combinatorial output.

***$AC1_n$ applies to these I/O pins only.

GAL20V8QS Logic Diagram



JEDEC Logic Array Cell Number = Product Line First Cell Number + Input Line Number

TL/L/11144-23

FIGURE 7

Programming Details

Understanding the information in this section is not essential when using approved programming equipment and software for developing GAL designs. This is a more thorough disclosure of the GAL architecture provided for direct JEDEC cell-map editing and diagnostic purposes. This section alone, however, does not contain sufficient information to implement the GAL programming algorithm. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

As mentioned in the Functional Description, the OLMC is responsible for selecting input and/or output paths, registered vs. combinatorial outputs, active-high or low polarity, and common vs. locally-controlled TRI-STATE control. Additionally, the OLMCs select between alternate logic array input paths to maintain JEDEC cell-map compatibility with either "small-PAL" or "medium-PAL" logic arrays.

The various configurations of the OLMCs are controlled by a set of programmable "architecture" cells, separate from the logic-defining array cells. Each GAL device contains two "global" architecture cells, "SYN" and "AC0", which affect all OLMCs. Each of the devices's eight OLMCs also contains two "local" cells, "AC1" and "XOR". The OLMC Logic Diagram in *Figure 6* shows how the architecture cells select the different paths through the OLMC.

The SYN bit controls whether a device will have any registered outputs (SYN = 0) or will be purely combinatorial (SYN = 1). The SYN bit determines whether device pins* 1 and 13 are used as the clock and global TRI-STATE control inputs (SYN = 0) or whether they are ordinary inputs (SYN = 1). The AC0 bit selects between the "Small-PAL" mode and the "Medium/Registered-PAL" modes. The function of the AC1 bits depend on the state of the AC0 bit. In "Small-PAL" mode (AC0 = 0), the AC1 bit in each OLMC determines whether the associated device pin is an output (AC1 = 0) or an input (AC1 = 1). In "Registered-PAL" mode (AC0 = 1), the AC1 bit determines whether each OLMC is registered (AC1 = 0) or combinatorial (AC1 = 1). In "Medium-PAL" mode (AC0 = 1), the AC1 bits in all OLMCs must be set to 1 (combinatorial). All of the valid architecture bit configurations are shown in the OLMC Architecture table (Table IV), which has the same familiar format used in the OLMC Selection table (Table I).

Independent of SYN, AC0 and the AC1 bits, the XOR bit in each OLMC selects between active-low (XOR = 0) or active-high (XOR = 1) output polarity.

*Applies to 24-pin DIP packages for GAL20V8QS; refer to the 28-lead PLCC Connection Diagram for conversion.

National Semiconductor

GAL22V10, -15, -20, -25, -30 Generic Array Logic

General Description

The NSC E²CMOS™ GAL® devices combine a high performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The 24-pin GAL22V10 features 22 inputs, and 10 programmable Output Logic Macro Cells (OLMCs) allowing each TRI-STATE® output to be configured by the user. The architecture of each output is user-programmable for registered or combinatorial operation, active high or low polarity, and as an input, output or bidirectional I/O. This architecture features variable product term distribution, from 8 to 16 logical product terms to each output, as shown in the logic diagram. CMOS circuitry allows the GAL22V10 to consume just 90 mA typical I_{CC} which represents a 50% saving in power when compared to its bipolar counterparts. Synchronous preset and asynchronous reset product terms have been added which are common to all output registers to enhance system operation. The GAL22V10 is directly compatible with the bipolar PAL22V10 in terms of functionality, fuse map, pinout, and electrical characteristics.

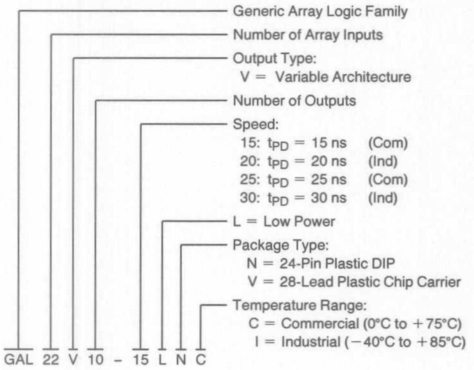
Programming is accomplished using industry standard available hardware and software tools. NSC guarantees a minimum 100 erase/write cycles.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, NSC guarantees 100% field programmability of all GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

Features

- ■ High performance E²CMOS technology
  - — 15 ns maximum propagation delay
  - — f_{max} = 45 MHz with feedback
  - — TTL compatible 16 mA outputs
  - — UltraMOS® III advanced CMOS technology
  - — Internal pull-up resistor on all pins
- ■ Electrically erasable cell technology
  - — Reconfigurable logic
  - — Reprogrammable cells
  - — 100% tested/guaranteed 100% yields
  - — High speed electrical erasure (<50 ms)
  - — 20 year data retention
- ■ Ten output logic macrocells
  - — Maximum Flexibility
  - — Programmable output polarity
  - — Maximum flexibility for complex logic designs
  - — Full function/fuse map/parametric compatibility with PAL22V10 devices
- ■ Variable product term distribution
  - — From 8 to 16 product terms per output data function
- ■ Global synchronous preset and asynchronous reset
- ■ Preload and power-up reset of all registers
  - — 100% functional testability
- ■ Fully supported by National OPAL™ and OPALjr development software
- ■ Security cell prevents copying logic

Ordering Information



Generic Array Logic Family

Number of Array Inputs

Output Type:
V = Variable Architecture

Number of Outputs

Speed:

15:	t_{PD} = 15 ns	(Com)
20:	t_{PD} = 20 ns	(Ind)
25:	t_{PD} = 25 ns	(Com)
30:	t_{PD} = 30 ns	(Ind)

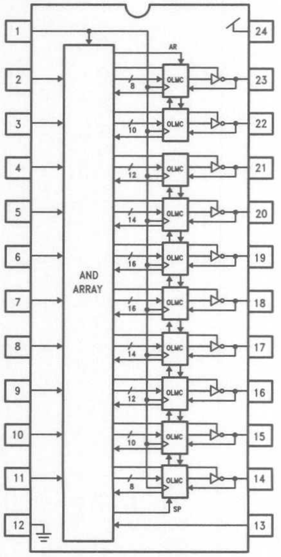
L = Low Power

Package Type:
N = 24-Pin Plastic DIP
V = 28-Lead Plastic Chip Carrier

Temperature Range:
C = Commercial (0°C to +75°C)
I = Industrial (−40°C to +85°C)

GAL 22 V 10 − 15 L N C

Block Diagram—GAL22V10



TL/L/10406–2

Absolute Maximum Ratings (Note 1)

Supply Voltage (V_{CC}) (Note 2)	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C

Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance C_{ZAP} = 100 pF R_{ZAP} = 1500Ω	700V

Test Method: Human Body Model
Test Specification: NSC SOP-5-028

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL22V10-15L		GAL22V10-20L		GAL22V10-25L		GAL22V10-30L		Units
			COM		IND		COM		IND		
			Min	Max	Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		12		15		15		25		ns
t_H	Hold Time (Input after Clock)		0		0		0		0		ns
t_W	Clock Pulse Width (High/Low)		8		10		15		20		ns
t_{AW}	Asynchronous Reset Input Pulse Width		15		20		25		30		ns
t_{AR}	Asynchronous Reset Recovery Time		15		20		25		30		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		22		27		30		45		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback		45.5		34.5		33.3		22.2	
		Without Feedback		62.5		50		33.3		25	MHz
f_I	Input Frequency (Note 5)			66.6		50.0		40.0		33.3	
t_{PR}	Clock Valid after Power-Up			100		100		100		100	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside the specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: f_{CLK} (with feedback) = $(t_{CYCLE})^{-1}$
f_{CLK} (without feedback) = $(2\ t_W)^{-1}$

Note 5: $f_I = (t_{PD})^{-1}$

Electrical Characteristics Over Recommended Operating Conditions

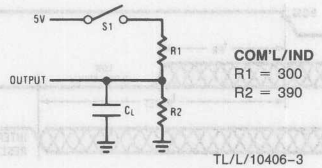
Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		$V_{CC}+1$	V
V_{IL}	Low Level Input Voltage				$V_{SS}-0.5$		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	$I_{OH} = -3.2$ mA	COM/IND	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 16 mA	COM/IND			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, $V_O = V_{CC}$ (Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−150	μA
I_I	Maximum Input Current	V_{CC} = Max, $V_I = V_{CC}$ (Max)			−150		10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, $V_I = V_{CC}$ (Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−150	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−150	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max		COM		90	130	mA
				IND			150	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					10	pF

*One output at a time for a maximum duration of one second.

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	GAL22V10-15L		GAL22V10-20L		GAL22V10-25L		GAL22V10-30L		Units
			COM		IND		COM		IND		
			Min	Max	Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	S1 Closed, C_L = 50 pF		15		20		25		30	ns
t_{CLK}	Clock to Registered Output or Feedback	S1 Closed, C_L = 50 pF		10		12		15		20	ns
t_{PZXI}	Input to Combinatorial Output Enabled via Product Term	Active High; S1 Open, C_L = 50 pF Active Low; S1 Closed, C_L = 50 pF		15		20		25		25	ns
t_{PXZI}	Input to Combinatorial Output Disabled via Product Term	From V_{OH}; S1 Open, C_L = 5 pF From V_{OL}; S1 Closed, C_L = 5 pF		15		20		25		25	ns
t_{AP}	Asynchronous Reset Input to Register Output			20		25		25		30	ns
t_{RESET}	Power-Up to Registered Output High	S1 Closed, C_L = 50 pF		45		45		45		45	μs

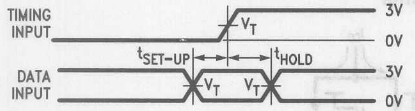
AC Test Load



TL/L/10406-3

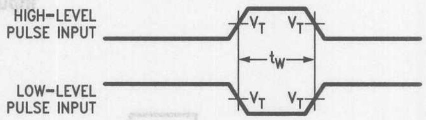
Test Waveforms

Setup and Hold



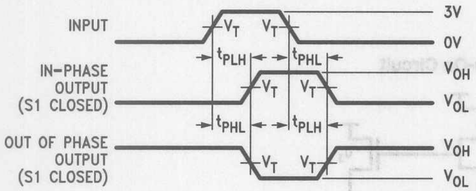
TL/L/10406-4

Pulse Width



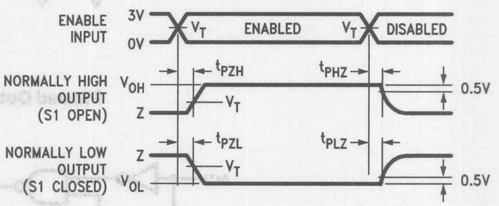
TL/L/10406-5

Propagation Delay



TL/L/10406-6

Enable and Disable



TL/L/10406-7

Notes:

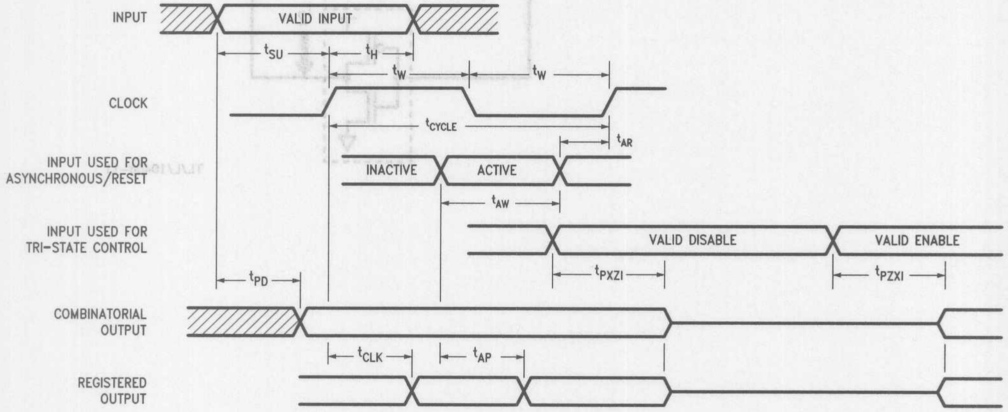
C_L includes probe and jig capacitance.

V_T = 1.5V.

Test inputs have rise and fall times of 3 ns 10%–90%.

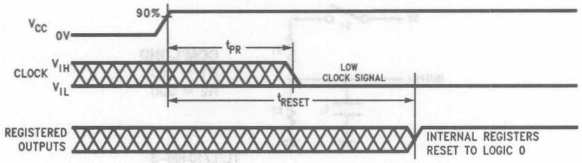
In the examples above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Switching Waveforms



TL/L/10406-8

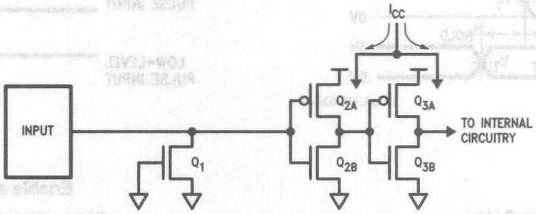
Power-Up Reset Waveforms



TL/L/10406–9

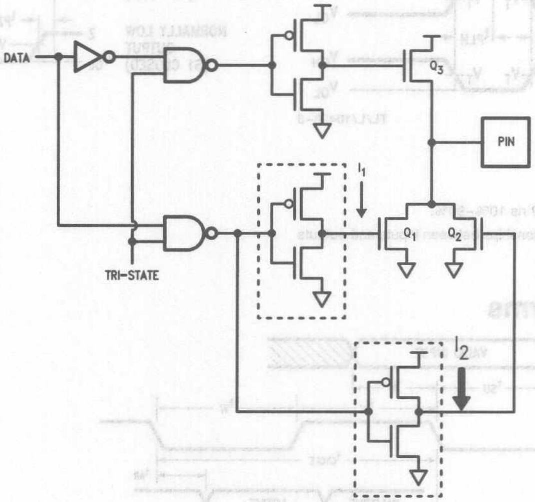
Input/Output Schematics

Input Translator/Buffer



TL/L/10406–22

Phased Output Turn-On Circuit



TL/L/10406–11

Functional Description

The GAL22V10 logic array consists of a programmable AND array with fixed OR-gate connections, similar to the traditional bipolar PAL architecture. The logic array is organized as 22 complementary input lines crossing 132 "product term" lines with a programmable E^{2}PROM cell at each intersection (5808 cells). Each programmable cell may establish a connection between an input line (true or complement phase of an array input signal) and a product term. A product term is satisfied (logically true) while all of the input lines "connected" to it are in the high logic state.

Of the 132 product terms, 130 are distributed among ten "output logic macrocells" (OLMCs) with a varying number of terms allocated to each OLMC (as shown in *Figure 1*). The ten OLMCs control the flow of input and output signals between the logic array and the device's I/O pins. For a given OLMC, 8, 10, 12, 14 or 16 product terms feed into an OR-gate to produce each output value. This varied distribution of product terms among outputs allows more optimum use of device resources. One additional product term in each of the ten OLMCs is used to control the associated TRI-STATE device output. One global product term is used to control an asynchronous preset, and another global product term is used for a synchronous reset, and both are connected to all ten of the output registers.

The fundamental transfer function of each GAL22V10 output is the familiar Boolean sum-of-products. Design development software is available which accepts Boolean equations and converts them automatically into GAL22V10 programming patterns.

Under control of an OLMC, each output may be designated either registered or combinatorial (non-registered). In the registered output configuration, the logic function output passes through a D-type flip-flop triggered by the rising edge of the clock input. Additionally, the logic function's output polarity may be designated active-low or active-high (adjusted after the register, if present). OLMC options such as these are selected using a set of programmable architecture control cells. These architecture cells are normally configured automatically by the development software or programming hardware.

The four possible I/O configurations of each GAL22V10 OLMC are: registered-active low, registered-active high, combinatorial-active low, and combinatorial-active high. These combinations are shown in *Figure 3*. The feedback paths are redirected with the register selection. The registered configurations include an internal feedback path taken directly from the register output. The combinatorial configurations include feedback from the I/O pin, thus allowing for bidirectional I/O or additional input channels.

All registers in a GAL22V10 device are reset to the low state upon power-up. Outputs, in turn, assume either low or high logic levels (if enabled) depending on the selected output polarity. Power-up reset may simplify sequential circuit design and test. To ensure successful power-up reset, V_{CC} must rise monotonically until the specified operating voltage is attained. During power-up, the clock input should be low as early as possible (within the specified time, t_{PR}) to avoid interfering with the reset operation. The clock input should also remain stable until after the power-up reset operation is completed to allow the registers to capture the proper next state on the first high-going clock transition.

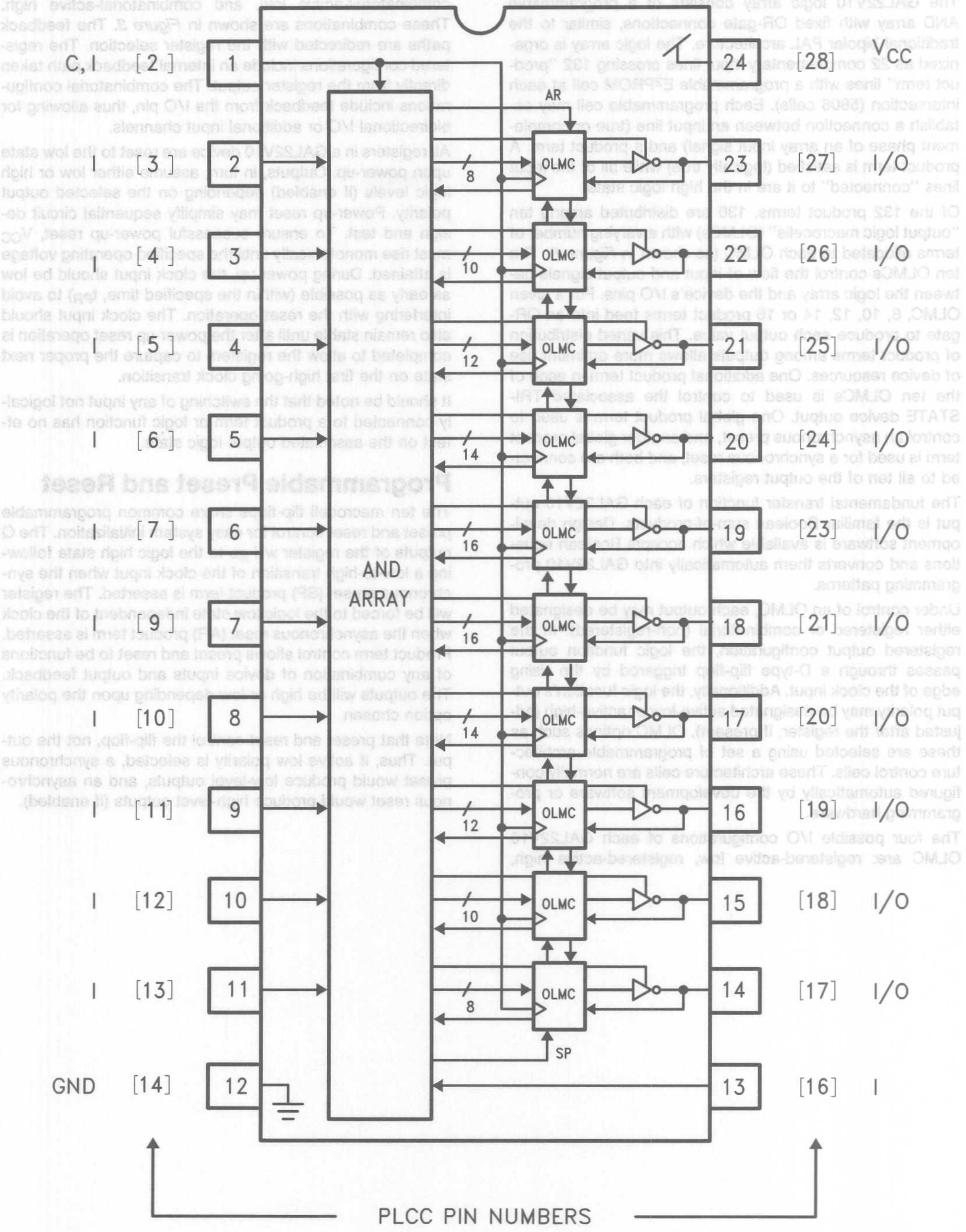
It should be noted that the switching of any input not logically connected to a product term or logic function has no effect on the associated output logic state.

Programmable Preset and Reset

The ten macrocell flip-flops share common programmable preset and reset control for easy system initialization. The Q outputs of the register will go to the logic high state following a low-to-high transition of the clock input when the synchronous preset (SP) product term is asserted. The register will be forced to the logic low state independent of the clock when the asynchronous reset (AR) product term is asserted. Product term control allows preset and reset to be functions of any combination of device inputs and output feedback. The outputs will be high or low depending upon the polarity option chosen.

Note that preset and reset control the flip-flop, not the output. Thus, if active low polarity is selected, a synchronous preset would produce low-level outputs, and an asynchronous reset would produce high-level outputs (if enabled).

GAL22V10 Block Diagram—DIP Connections



TL/L/10406-12

PCC Pin Numbers
FIGURE 1

28-Lead PLCC Connection Diagram

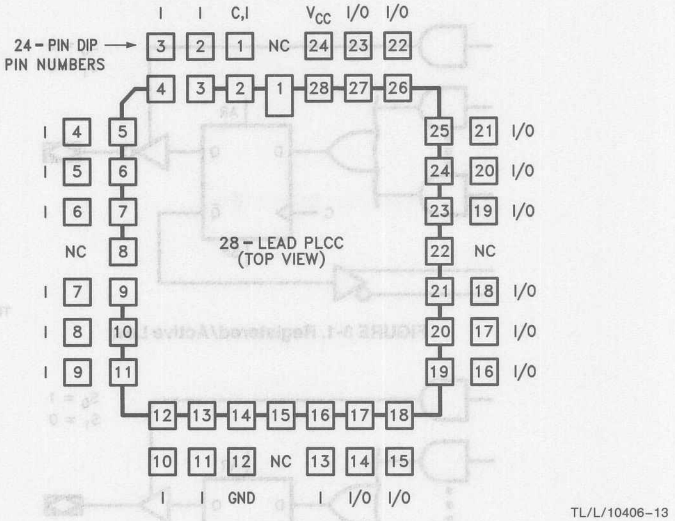


FIGURE 2

Clock/Input Frequency Specifications

The clock frequency (f_{CLK}) parameter listed in the Recommended Operating Conditions table specifies the maximum speed at which the GAL22V10 registers are guaranteed to operate. Clock frequency is defined differently for the two cases in which register feedback is used versus when it is not. In a data-path type application, when the logic functions fed into the registers are not dependent on register feedback from the previous cycle (i.e. based only on external inputs), the minimum required cycle period (f_{CLK}^{-1} without feedback) is defined as the greater of the minimum clock period (t_w high + t_w low) and the minimum "data window" period (t_{SU} + t_H). This assumes optimal alignment between data inputs and the clock input. In sequential logic applications such as state machines, the minimum required cycle period ($t_{CYCLE} = f_{CLK}^{-1}$ with feedback) is defined as t_{CLK} + t_{SU}. This provides sufficient time for outputs from the registers to feed back through the logic array and set up on the inputs to the registers before the end of each cycle.

The input frequency (f_I) parameter specifies the maximum rate at which each GAL22V10 input can be toggled and still produce valid logic transitions on each combinatorial output. The f_I specification is derived as the inverse of the combinatorial propagation delay (t_{PD}).

Design Development Support

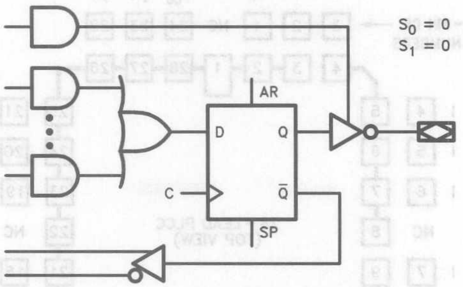
A variety of software tools and programming equipment are available to support the development of designs using GAL22V10 products. Typical software packages, including National's OPAL software, accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate a JEDEC-compatible "cell-map" (analogous to a PAL "fuse-map"). The industry-standard JEDEC format ensures that the resulting cell-map file can be down-loaded into industry standard programming equipment. Many software packages and programming units support a multitude of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

National strongly recommends using only approved programming hardware and software for developing GAL designs. Programming using unapproved equipment generally voids all guarantees. Approved programmers incorporate specialized programming algorithms that program the array and automatically configure the architecture cells. To ensure data retention and reliability, the programming algorithm also tracks the number of programming cycles to which each GAL device has been subjected since shipment, and stores this information automatically in the device.

The GAL22V10 can accept fuse-maps prepared for other PAL22V10 devices. PAL22V10 fuse-maps can be created by any JEDEC-compatible PAL development software or by loading the fuse pattern from an existing programmed PAL22V10 device into the programming unit (provided the PAL device has not been secured).

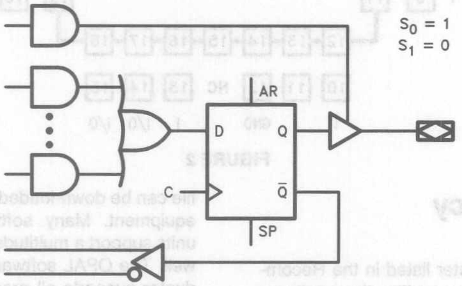
Detailed logic diagrams showing all JEDEC cell-map addresses in the GAL22V10 logic array and OLMC are provided for direct map editing and diagnostic purposes. *Figure 6* and Table II show details of the OLMC and the programmable architecture cell combinations. *Figure 7* shows the JEDEC logic diagram and details of all programmable cell locations. For a list of current software and programming support tools available for these devices, please contact your local National sales representative or distributor. If detailed specifications of the GAL22V10 programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

OLMC Selection Table



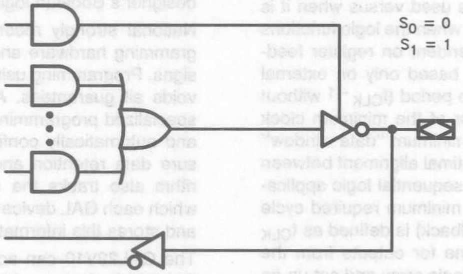
TL/L/10406–14

FIGURE 3-1. Registered/Active Low



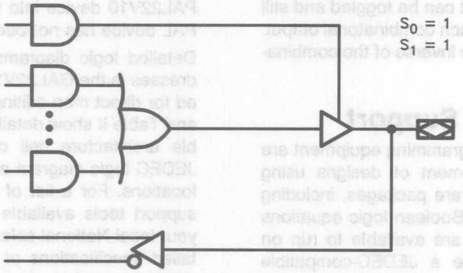
TL/L/10406–15

FIGURE 3-2. Registered/Active High



TL/L/10406–16

FIGURE 3-3. Combinatorial/Active Low



TL/L/10406–17

FIGURE 3-4. Combinatorial/Active High

cess is disabled, preventing further programming or verification of the array. The security cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed.

Electronic Signature

Each GAL device contains an electronic signature word consisting of 64 bits of reprogrammable memory. The electronic signature word can be programmed to contain any identification information desired by the user. Some uses include pattern identification labels, revision numbers, dates, inventory control information, etc. The data stored in the electronic signature word has no effect on the functionality of the device. The information is read out of the device using the normal program verification procedure provided by the programming equipment. The information may be accessed at any time independent of the state of the security cell. National's OPAL development software allows electronic signature data to be entered by the user and downloaded to the programming equipment.

Bulk Erase

The programming equipment automatically performs a bulk erase operation prior to each programming operation. No special erase operation need be performed by the user. Bulk erase clears the logic array, architecture cells, security cell, and electronic signature information. The GAL device is thereby reverted back to its virgin state.

Latch-Up Protection

GAL devices are designed with an on-chip charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

Manufacturer Testing

Because of E^{2}CMOS technology, GAL devices can be reprogrammed in milliseconds. This allows each device to be completely tested by the manufacturer using numerous logic array and architecture patterns prior to shipping. Every programmable cell and every logic path through every device is fully tested for programmability, functionality and performance to all AC and DC parameters. The customer can therefore expect 100% programming and functional yield and 100% compliance of all GAL products to datasheet specifications.

The testing procedure performed on all GAL devices by the manufacturer tests all aspects of device operation. Extensive testing of all programmable cells in the device include margin testing, internal verify, and program retention during tions.

Register Preload

The register preload feature allows OLMC registers to be directly loaded with any desired data pattern. It also allows the present state of OLMC registers to be examined regardless of TRI-STATE control conditions. This simplifies testing of devices after programming. A device may be put into any desired register state at any point during the functional test sequence. The test sequence may then be resumed to verify proper next-state transitions. This allows complete verification of sequential logic circuits, including states that are normally impossible or difficult to reach. It may also shorten the overall test time significantly.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. To verify these transitions requires the ability to set the state registers into an arbitrary "present state" value, and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is then clocked into a new state, or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Register preload is not an operational mode and is not intended for board-level testing because elevated voltage levels must be applied to the device. The programming equipment normally provides the register preload capability as part of its functional test facility. Note that the testing of GAL devices after programming by the user may be considered unnecessary because all E^{2}CMOS GAL products are completely tested by the manufacturer, guaranteeing 100% post-programming functional yield.

The register preload algorithm is described for those users who wish to test programmed GAL devices using test equipment other than approved GAL programming equipment. As shown in the register preload waveform in *Figure 5,* the preload sequence must not begin until the normal power-up reset operation has completed (after time t_{RESET}). The device is placed into preload mode by raising the "PRLD" input (pin 13*) to voltage V_{IES}, as specified in the register preload specifications (Table I).

To preload the OLMC registers, a series of data bits are shifted into the device on the "S_{DIN}" input (pin 11*), one bit for each OLMC in which registered output has been selected. (Non-registered OLMCs are bypassed.) The shift sequence is clocked by the rising edge of the "D_{CLK}" input (pin 1*). The data stream is shifted in through the registered OLMC with the lowest corresponding pin number, and then "upward" through all remaining registered OLMCs in pin-number ascending order. Therefore, the first data bit in the series is ultimately loaded into the registered OLMC with the highest corresponding pin number, as shown in *Figure 4.*

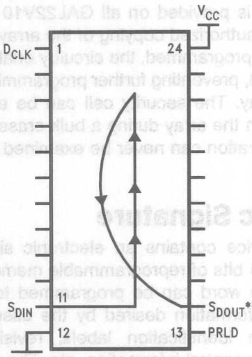
*Applies to 24-pin DIP packages for GAL22V10; refer to the 28-lead PCC Connection Diagram for conversion.

Register Preload (Continued)

As the data series is shifted into the S_{DIN} input, the contents of all registers (in registered OLMCs) are shifted "upward" and out onto the "S_{DOUT}" output (pin 14*). Complete present-state information can be examined in this manner. Test fixtures can be devised to test several GAL devices in which the S_{DOUT} pin of each chip is connected to the S_{DIN} pin of the next, and all preload and present-state data can be shifted around a single serial loop.

Note that when shifting register data into S_{DIN} or out of S_{DOUT}, V_{IL}/V_{OL} = register reset (0), and V_{IH}/V_{OH} = register set (1). These 0 and 1 register states are always inverted (active-low) on the normal output pins regardless of the selected output polarity (polarity affects logic function values before register inputs).

*Applies to 24-pin DIP packages for GAL22V10; refer to the 28-lead PCC Connection Diagram for conversion.



TL/L/10406–18

**The S_{DOUT} output buffer is an open drain output during preload. This pin should be terminated to V_{CC} with a 10 kΩ resistor.

FIGURE 4. Output Register Preload Pinout

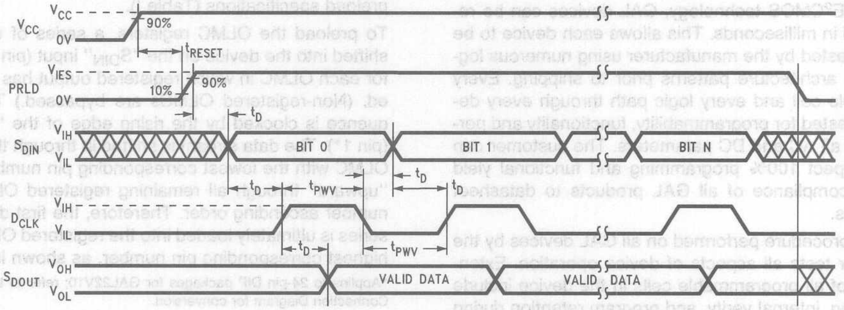
Register Preload Specifications

TABLE I

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	Input Voltage (High)		2.40		V_{CC}	V
V_{IL}	Input Voltage (Low)		0.00		0.50	V
V_{IES}	Register Preload Input Voltage		14.5	15	15.5	V
V_{OH}	Output Voltage (High) (Note 1)				V_{CC}	V
V_{OL}	Output Voltage (Low) (Note 1)	$I_{OL} \leq 12$ mA	0.00		0.50	V
I_{IH}, I_{IL}	Input Current (Programming)			±1	±10	μA
I_{OH}	High Level Output Current (Note 1)	$V_{OH} \leq V_{CC}$			10	μA
t_{PWV}	Verify Pulse Width		1	5	10	μs
t_D	Pulse Sequence Delay			5	10	μs
t_{RESET}	Register Reset Time from Valid V_{CC}				45	μs

Note 1: The S_{DOUT} output buffer is an open drain output. This pin should be terminated to V_{CC} with a 10k resistor.

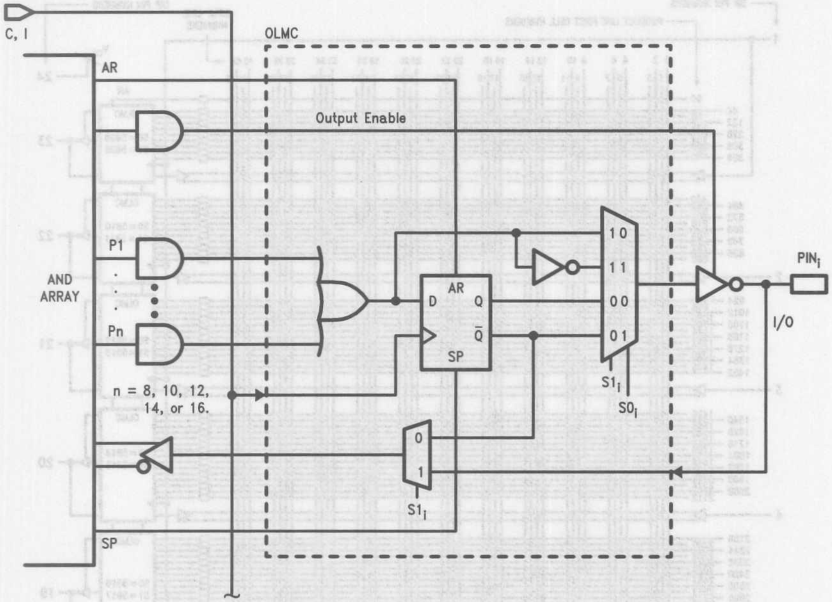
Register Preload Waveforms



TL/L/10406–19

FIGURE 5

OLMC Logic Diagram



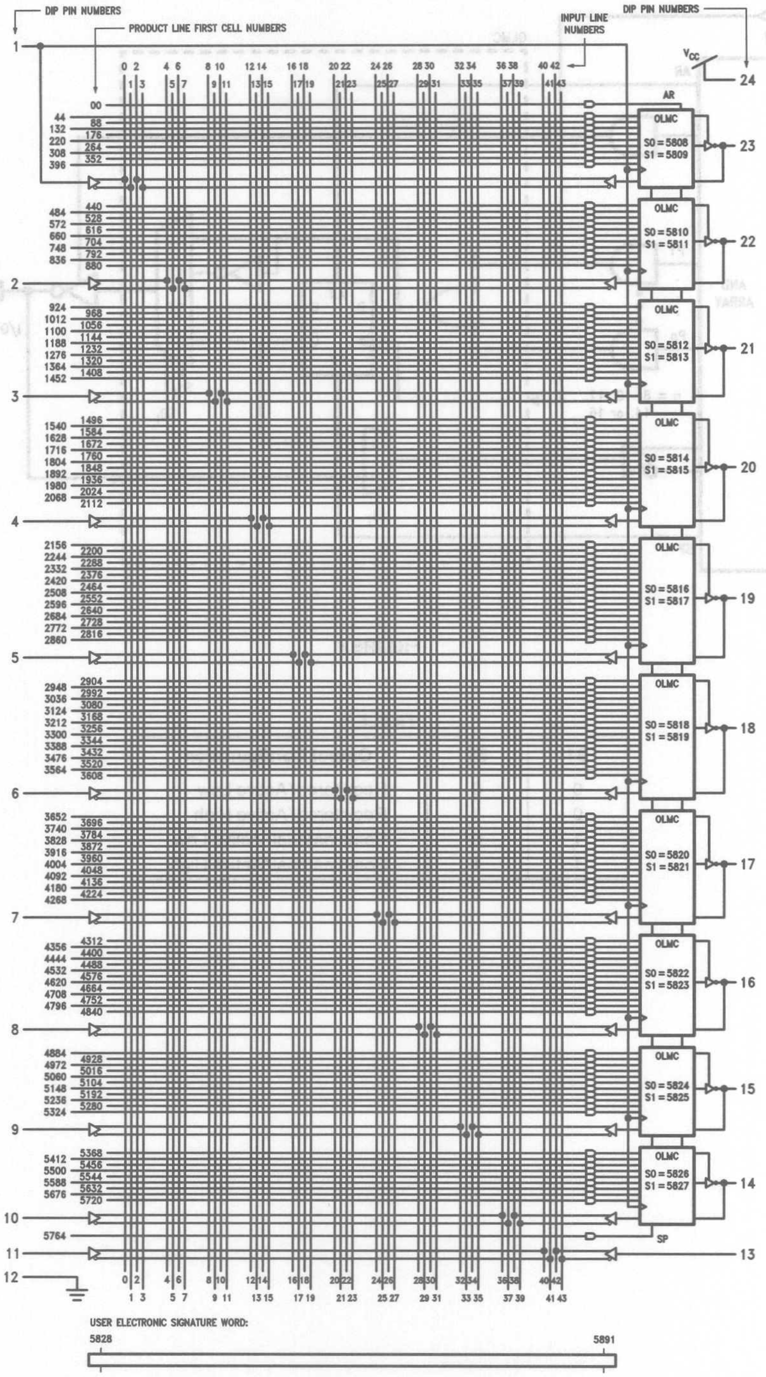
TL/L/10406-20

FIGURE 6

TABLE II

S1	S0	Output Configuration
0	0	Registered/Active Low
0	1	Registered/Active High
1	0	Combinatorial/Active Low
1	1	Combinatorial/Active High

GAL22V10 Logic Diagram



TL/L/10406-21

JEDEC Logic Array Cell Numbers = Product Line First Cell Number + Input Line Numbers

FIGURE 7

National Semiconductor

GAL20RA10-15, -20, -25 Generic Array Logic

General Description

The NSC E^{2}CMOS™ GAL® device combines a high performance CMOS process with electrically erasable floating gate technology. This programmable memory technology applied to array logic provides designers with reconfigurable logic and bipolar performance at significantly reduced power levels.

The GAL20RA10 is made up of ten Output Logic Macro Cells (OLMC). Four programmable AND array outputs feed into the fixed OR-gate for each OLMC to generate the device's output functions. Four other AND array outputs are used for control functions in the OLMC. With a robust mixture of logic derived controlled functions and selectable output data paths, the GAL20RA10 provides an ideal solution for registered random logic applications.

This device is housed in a 24-pin 300 mil DIP. A 28-pin PCC package is also available. It can be programmed by most PAL programmers.

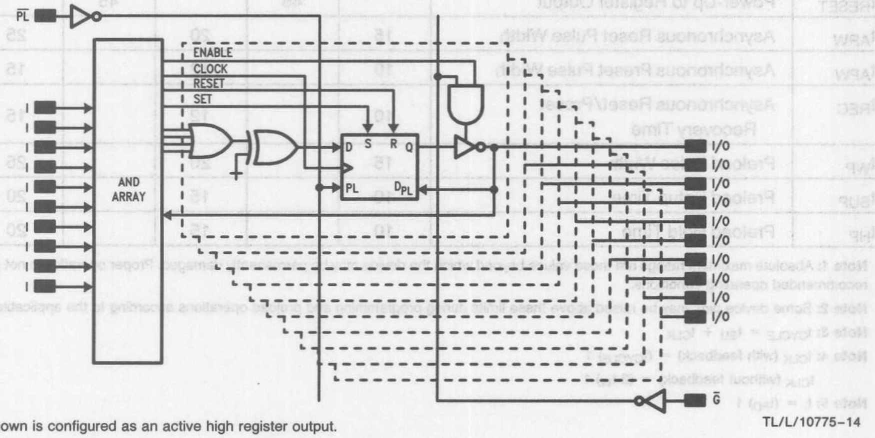
Programming is accomplished using industry standard available hardware and software tools. NSC guarantees a minimum 100 erase/write cycles.

Unique test circuitry and reprogrammable cells allow complete AC, DC, cell and functionality testing during manufacture. Therefore, NSC guarantees 100% field programmability of the GAL devices. In addition, electronic signature is available to provide positive device ID. A security circuit is built-in, providing proprietary designs with copy protection.

Features

- High performance E^{2}CMOS technology
  - — 15 ns maximum propagation delay
  - — f_{CLK} = 45 MHz
  - — 15 ns maximum from clock input to data output
  - — TTL compatible 16 mA outputs
  - — UltraMOS® III advanced CMOS technology
- Electrically erasable cell technology
  - — Reconfigurable logic
  - — Reprogrammable cells
  - — 100% tested/guaranteed 100% yields
  - — High speed electrical erasure (<50 ms)
  - — 20 year data retention
- 10 output logic macrocells
  - — Maximum flexibility for complex logic designs
  - — Programmable output polarity
  - — Programmable asynchronous set and reset
  - — Individually programmable clocks
  - — Programmable and dedicated pin control of output TRI-STATE®
  - — Programmable Register bypass
  - — TTL level Register preload
- Power-up reset for registered outputs
- JEDEC-compatible programming equipment and development software available
- Preload and power-up reset of all registers
  - — 100% functional testability
- Full supported development software
- Electronic signature for identification
- Security fuse prevents direct copying of logic patterns
- JEDEC map identicle to Bipolar PAL versions
- Fully supported by National OPAL™ and OPALjr development software

Block Diagram—GAL20R10



*Output macrocell shown is configured as an active high register output.

TL/L/10775–14

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	550V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C

AC TIMING REQUIREMENTS

Symbol	Parameter		GAL20RA10-15		GAL20RA10-20		GAL20RA10-25		Units
			COM		IND		COM		
			Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock		7		10		15		ns
t_H	Hold Time (Input after Clock)		0		0		0		ns
t_W	Clock Pulse Width (High/Low)		10		12		15		ns
t_{CYCLE}	Clock Cycle Period (with Feedback) (Note 3)		22		30		40		ns
f_{CLK}	Clock Frequency (Note 4)	With Feedback	45		33		25.0		MHz
		Without Feedback	50.0		41.7		33.3		MHz
f_I	Input Frequency (Note 5)		66.7		50.0		40.0		MHz
t_{PR}	Clock Valid after Power-Up		100		100		100		ns
t_{RESET}	Power-Up to Register Output			45		45		45	μs
t_{ARW}	Asynchronous Reset Pulse Width		15		20		25		ns
t_{APW}	Asynchronous Preset Pulse Width		10		12		15		ns
t_{REC}	Asynchronous Reset/Preset Recovery Time		10		12		15		ns
t_{WP}	Preload Pulse Width		15		20		25		ns
t_{SUP}	Preload Setup Time		10		15		20		ns
t_{HP}	Preload Hold Time		10		15		20		ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Note 3: $t_{CYCLE} = t_{SU} + t_{CLK}$

Note 4: t_{CLK} (with feedback) = (t_{CYCLE}) 1
t_{CLK} (without feedback) = (2 t_W) 1

Note 5: t_I = (t_{PD}) 1

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions	Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage			2.0		V_{CC} + 1	V
V_{IL}	Low Level Input Voltage			−1.0		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = **Min** I_{OH} = −3.2 mA	COM/IND	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = **Min** I_{OL} = 8 mA	COM/IND			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = **Max**, V_O = V_{CC} **Max**				10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = **Max**, V_O = **GND**				−10	μA
I_I	Maximum Input Current	V_{CC} = **Max**, V_I = V_{CC} **Max**				10	μA
I_{IH}	High Level Input Current	V_{CC} = **Max**, V_I = V_{CC} **Max**				10	μA
I_{IL}	Low Level Input Current	V_{CC} = **Max**, V_I = **GND**				−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = **GND**		−30		−150	mA
I_{CC}	Supply Current	f = 15 **MHz**, V_{CC} = **Max**	COM			100	mA
			IND			120	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V				8	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V				10	pF

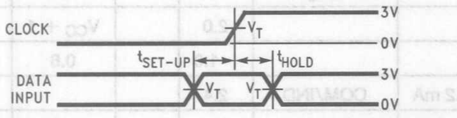
*One output at a time for a maximum duration of one second

Switching Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Test Conditions	GAL20RA10-15		GAL20RA10-20		GAL20RA10-25		Units
			COM		IND		COM		
			Min	Max	Min	Max	Min	Max	
t_{PD}	Input or Feedback to Combinatorial Output	C_L = 50 pF, S1 Closed		15		20		25	ns
t_{CLK}	Clock Input to Registered Output or Feedback	C_L = 50 pF, S1 Closed		15		20		25	ns
t_S	Asynchronous Set Input to Registered Output Low			15		20		25	ns
t_R	Asynchronous Reset Input to Registered Output High			15		20		25	ns
t_{PZXG}	$\overline{G}$ Pin Output Enabled	C_L = 50 pF, Active High: S1 Open, Active Low: S1 Closed		12		15		20	ns
t_{PXZG}	$\overline{G}$ Pin Output Disabled	C_L = 5 pF, From V_{OH}: S1 Open, From V_{OL}: S1 Closed		12		15		20	ns
t_{PZXI}	Input to Output Enabled via Product Term	C_L = 50 pF, Active High: S1 Open, Active Low: S1 Closed		15		20		25	ns
t_{PXZI}	Input to Output Disabled via Product Term	C_L = 5 pF, From V_{OH}: S1 Open, From V_{OL}: S1 Closed		15		20		25	ns

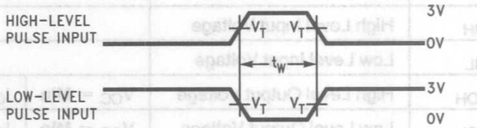
Test Waveforms

Set-Up and Hold



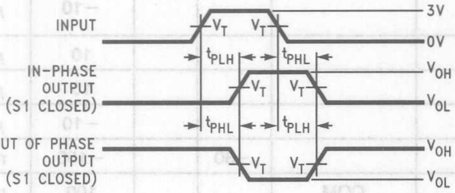
TL/L/10775-18

Pulse Width



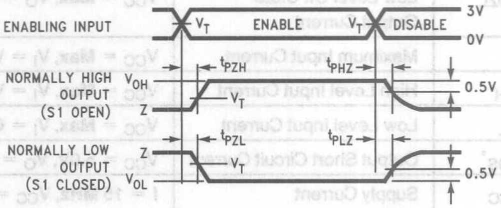
TL/L/10775-20

Propagation Delay



TL/L/10775-19

Enable and Disable



TL/L/10775-21

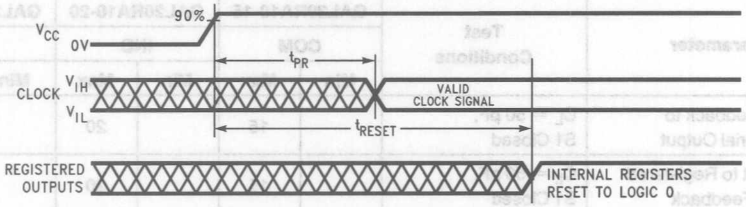
Notes:

$V_T = 1.5V$

C_L includes probe and jig capacitance.

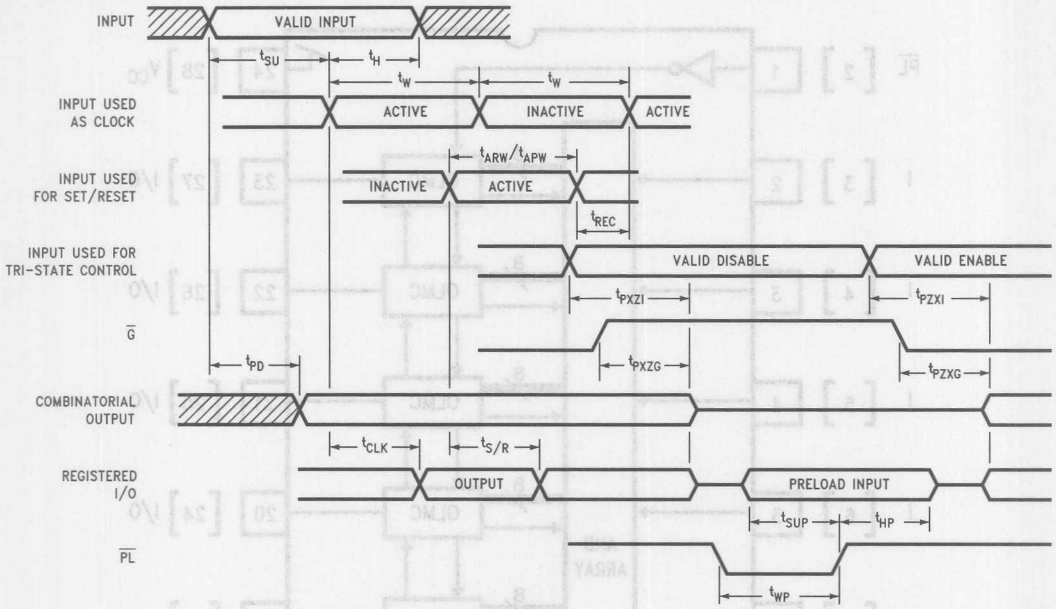
In the examples above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Power-Up Reset Waveforms



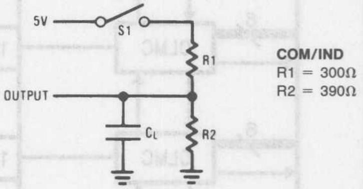
TL/L/10775-22

Switching Waveforms



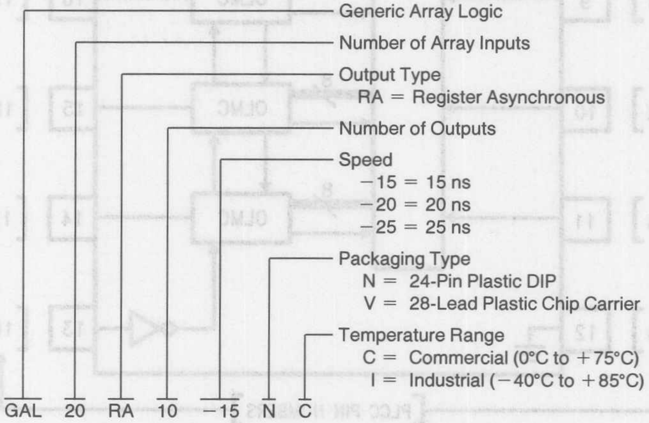
TL/L/10775-23

AC Test Load



TL/L/10775-5

Ordering Information



Generic Array Logic

Number of Array Inputs

Output Type
RA = Register Asynchronous

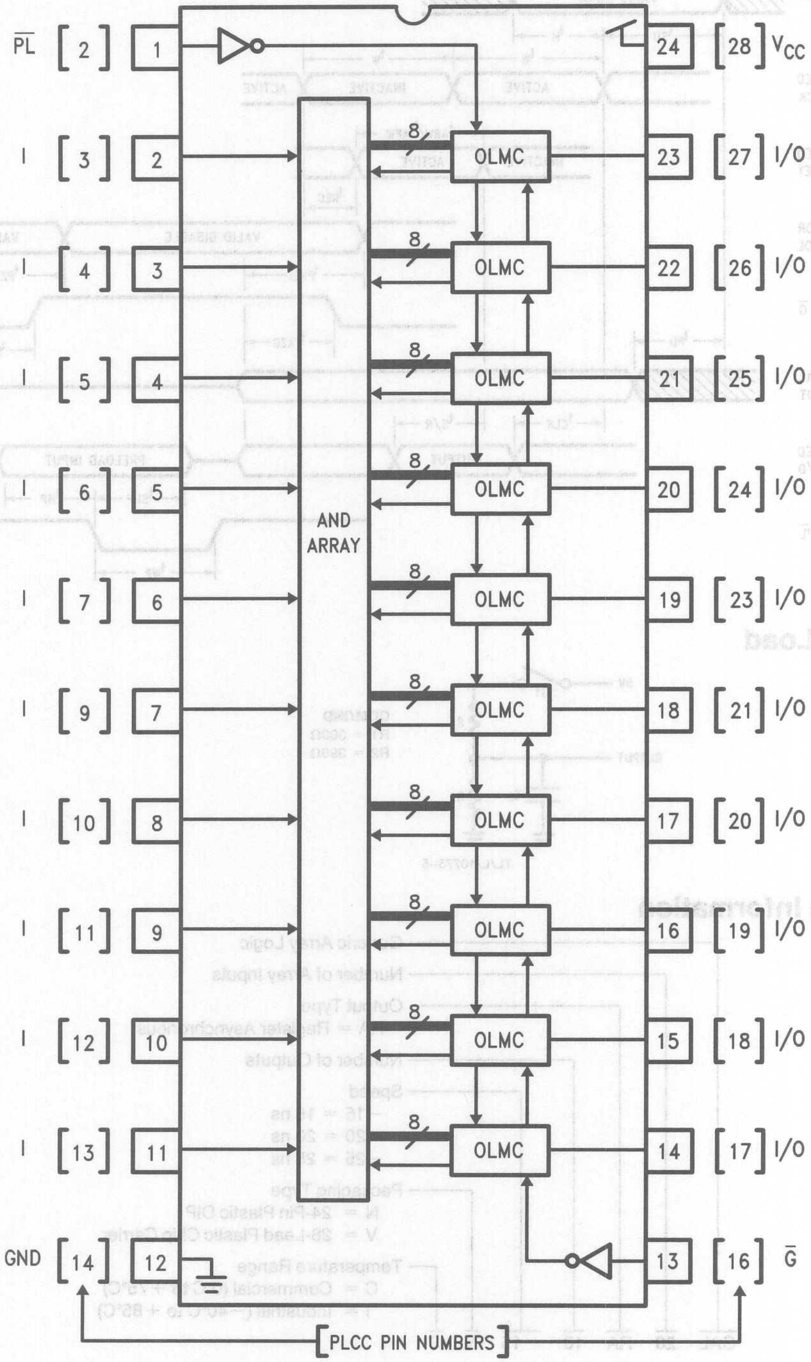
Number of Outputs

Speed
−15 = 15 ns
−20 = 20 ns
−25 = 25 ns

Packaging Type
N = 24-Pin Plastic DIP
V = 28-Lead Plastic Chip Carrier

Temperature Range
C = Commercial (0°C to +75°C)
I = Industrial (−40°C to +85°C)

GAL20RA10 Block Diagram—DIP Connections



TL/L/10775-3

terms are organized into ten groups of eight each. Four of the eight product terms in each group connect into an OR-gate to produce the sum-of-products logic function. The remaining four product terms in each group are used for control functions in the "Output Logic Macro Cell" (OLMC).

As shown in the GAL20RA10 Block Diagram a total of ten output logic functions are available. Under control of an OLMC each output may be designated either as a registered output configuration or combinatorial.

The logic function output passes through a D-type Flip-Flop triggered by the rising edge of the clock which is defined by one product-term line.

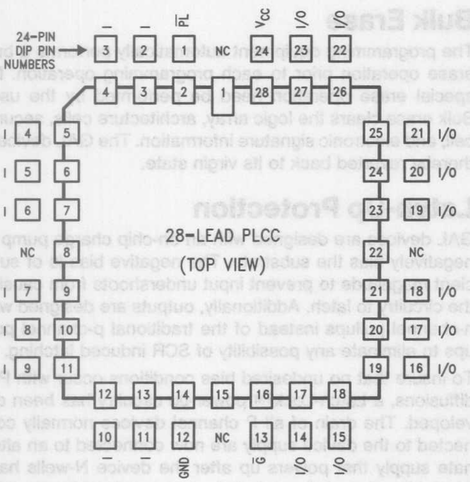
Two product-terms are designated to set or reset the output register and to define the output configuration (register or combinatorial).

Set	Reset	Output Mode
0	0	Register Mode
0	1	Reset
1	0	Set
1	1	Combinatorial Mode

All architecture cells are normally configured automatically by the development software.

Connection Diagram

28-Lead PLCC



TL/L/10775-4

PROGRAMMABLE SET AND RESET

In each OLMC cell, two product lines are dedicated to asynchronous set and reset. If the set product line is high, the register output becomes a logic "1", the output pin becomes a "0". If the reset product line is high, the register output becomes a logic "0", the output pin becomes a "1". The operation of the programmable set and reset overrides the clock.

lows each output to be configured in the registered or combinatorial mode.

PROGRAMMABLE CLOCK

One of the product lines in each group is connected to the clock. This provides the user with the additional flexibility of a programmable clock, so each output can be clocked independently of all the others.

PROGRAMMABLE AND HARD-WIRED TRI-STATE OUTPUTS

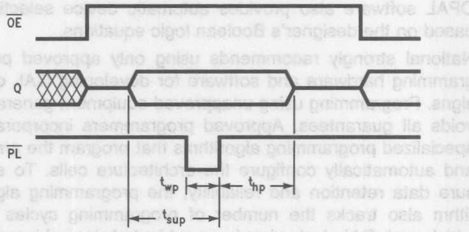
The GAL20RA10 provides a product term dedicated to output control. There is also an output control pin (Pin 13). The output is enabled if both the output control pin is low and the output control product term is HIGH. If the output control pin is high all outputs will be disabled or if an output control product term is low, then that output will be disabled.

PROGRAMMABLE OUTPUT POLARITY

The outputs can be programmed either active-low or active-high. This is represented by the exclusive-OR gates shown in the GAL20RA10 Logic Diagram. When the output polarity is unprogrammed the lower input to the exclusive-OR gate is high, so the output is active-high. Similarly, when the output polarity cell is 0, or a low impedance connection to GND, the output is active-low. The programmable output polarity features allows the user a high degree of flexibility when writing equations.

Output Register Preload

Register preload allows any arbitrary state to be loaded into the PAL output registers. This allows complete logic verification, including states that are impossible or impractical to reach. To use the preload feature, first disable the outputs by bringing $\overline{OE}$ high, and present the data at the output pins. A low-level on the preload pin ($\overline{PL}$) will then load the data into the registers.



TL/L/10775-24

POWER-UP RESET

The GAL20RA10 device resets all registers to a low state upon power-up (active-low outputs assume high logic levels if enabled). This may simplify sequential circuit design and test. During power-up, the clock input should assume a valid, stable logic state as early as possible to avoid interfering with the reset operation. The clock input should remain stable until after the power-up reset operation is completed to allow the registers to capture the proper next state on the first high-going transition.

Clock/Input Frequency Specifications

The clock frequency (f_{CLK}) parameter listed in the Recommended Operating Conditions table specifies the maximum speed at which the GAL registers are guaranteed to operate. Clock frequency is defined differently for the two cases in which register feedback is used versus when it is not. In a data-path type application, when the logic functions feed into the registers are not dependent on register feedback from the previous cycle (i.e., based only on external inputs), the minimum required cycle period (f_{CLK}^{-1} without feedback) is defined as the greater of the minimum "data window" period (t_W high + t_W low) and the minimum "data window" period ($t_{SU} + t_H$). This assumes optimal alignment between data inputs and the clock input. In sequential logic applications such as state machines, the minimum required cycle period ($t_{CYCLE} = f_{CLK}^{-1}$ with feedback) is defined as $t_{CLK} + t_{SU}$. This provides sufficient time for outputs from the registers to feed back through the logic array and set up on the inputs to the registers before the end of each cycle.

The input frequency (f_I) parameter specifies the maximum rate at which each GAL input can be toggled and still produce valid logic transitions on each combinatorial output. The f_I specification is derived as the inverse of the combinatorial propagation delay (t_{PD}).

Design Development Support

A variety of software tools and programming equipment is available to support the development of designs using GAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate a JEDEC-compatible "cell-map" (analogous to a PAL "fuse-map"). The industry-standard JEDEC format ensures that the resulting cell-map file can be down-loaded into a variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL™ software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

National strongly recommends using only approved programming hardware and software for developing GAL designs. Programming using unapproved equipment generally voids all guarantees. Approved programmers incorporate specialized programming algorithms that program the array and automatically configure the architecture cells. To ensure data retention and reliability, the programming algorithm also tracks the number of programming cycles to which each GAL device has been subjected since shipment, and stores this information automatically in the device.

The special GAL programming algorithm can also program a GAL device using a standard fuse-map developed for any of the emulated PAL products. PAL fuse-maps can be created by any JEDEC-compatible PAL development software or by loading the fuse pattern from an existing programmed PAL device into the programming unit (provided the PAL device has not been secured). However, to utilize the full flexibility of the GAL architecture, true GAL development software (such as OPAL software) is recommended.

Detailed logic diagrams showing all JEDEC cell-map addresses in the GAL logic array and OLMC are provided for direct map editing and diagnostic purposes (see "Programming Details"). For a list of current software and programming support tools available for these devices, please contact your local National sales representative or distributor. If detailed specifications of the GAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support department.

Security Cell

A security cell is provided on all GAL20RA10 devices as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, the circuitry enabling array access is disabled, preventing further programming or verification of the array. The security cell can be erased only in conjunction with the array during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed.

Electronic Signature

Each GAL device contains an electronic signature word consisting of 64 bits of reprogrammable memory. The electronic signature word can be programmed to contain any identification information desired by the user. Some uses include pattern identification labels, revision numbers, dates, inventory control information, etc. The data stored in the electronic signature word has no effect on the functionality of the device. The information is read out of the device using the normal program verification procedure provided by the programming equipment. The information may be accessed at anytime independent of the state of the security cell. National's OPAL development software allows electronic signature data to be entered by the user and downloaded to the programming equipment.

Bulk Erase

The programming equipment automatically performs a bulk erase operation prior to each programming operation. No special erase operation need be performed by the user. Bulk erase clears the logic array, architecture cells, security cell, and electronic signature information. The GAL device is thereby reverted back to its virgin state.

Latch-Up Protection

GAL devices are designed with an on-chip charge pump to negatively bias the substrate. The negative bias is of sufficient magnitude to prevent input undershoots from causing the circuitry to latch. Additionally, outputs are designed with n-channel pullups instead of the traditional p-channel pullups to eliminate any possibility of SCR induced latching.

To insure that no undesired bias conditions occur with P+ diffusions, a Latch-Lock™ power-up circuitry has been developed. The drain of all P channel devices normally connected to the device supply are now connected to an alternate supply that powers up after the device N-wells have been biased and the substrate has reached its negative clamp value. This prevents any hazardous bias conditions from developing in the power-up sequence. After power-up is complete, the Latch-Lock circuitry becomes dormant until a full power-down has occurred; this eliminates the chance of an unwanted P channel power-down during device operation.

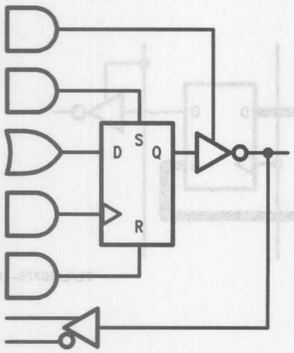
Manufacturer Testing

Because of E^2CMOS technology, GAL devices can be reprogrammed in milliseconds. This allows each device to be completely tested by the manufacturer using numerous logic array and architecture patterns prior to shipping. Every programmable cell and every logic path through every device is fully tested for programmability, functionality and performance to all AC and DC parameters. The customer can therefore expect 100% programming and functional yield and 100% compliance of all GAL products to data sheet specifications.

The testing procedure performed on all GAL devices by the manufacturer tests all aspects of device operation. Extensive testing of all programmable cells in the device include margin testing, internal verify, and program retention during high-temperature bake. All DC and AC parameters are tested at hot and cold temperatures using a variety of worstcase logic and signal patterns. Functional test include reprogramming each OLMC to all valid architectural configurations.

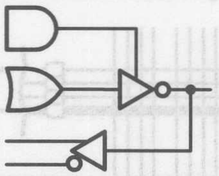
OLMC Configurations

Registered/Active Low



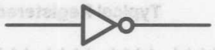
TL/L/10775-6

Combinatorial/Active Low



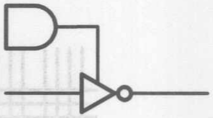
TL/L/10775-7

Output Always Enabled



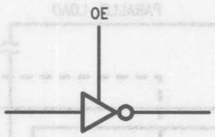
TL/L/10775-8

Programmable



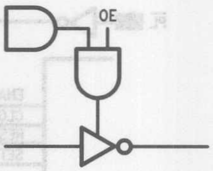
TL/L/10775-9

Hard-Wired



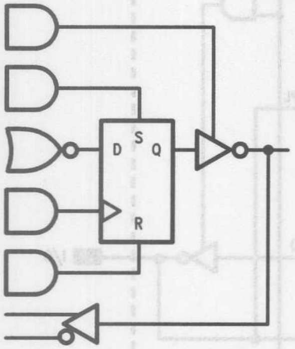
TL/L/10775-10

Combination of Programmable and Hard-Wired



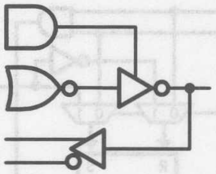
TL/L/10775-11

Registered/Active High



TL/L/10775-12

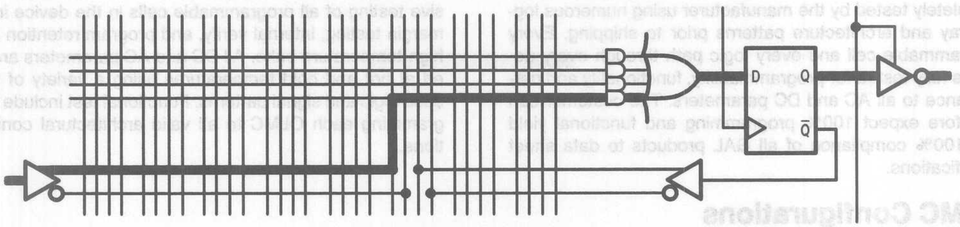
Combinatorial/Active High



TL/L/10775-13

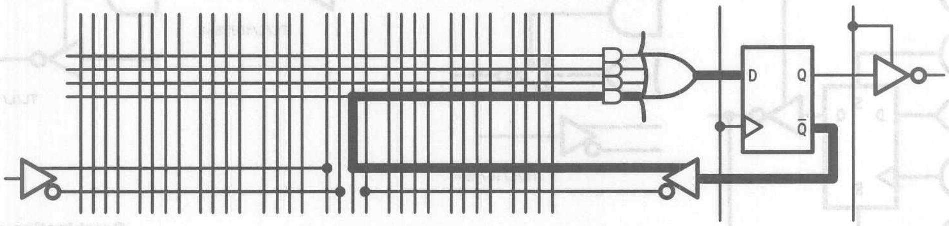
Functional Description

Typical Registered Logic Function Without Feedback

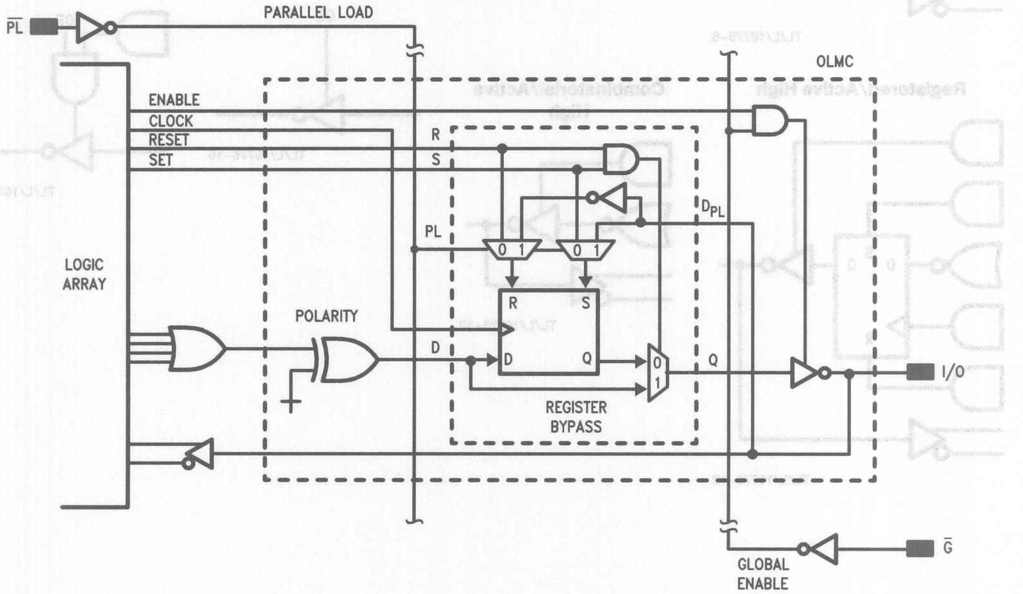


TL/L/10775-15

Typical Registered Logic Function With Feedback



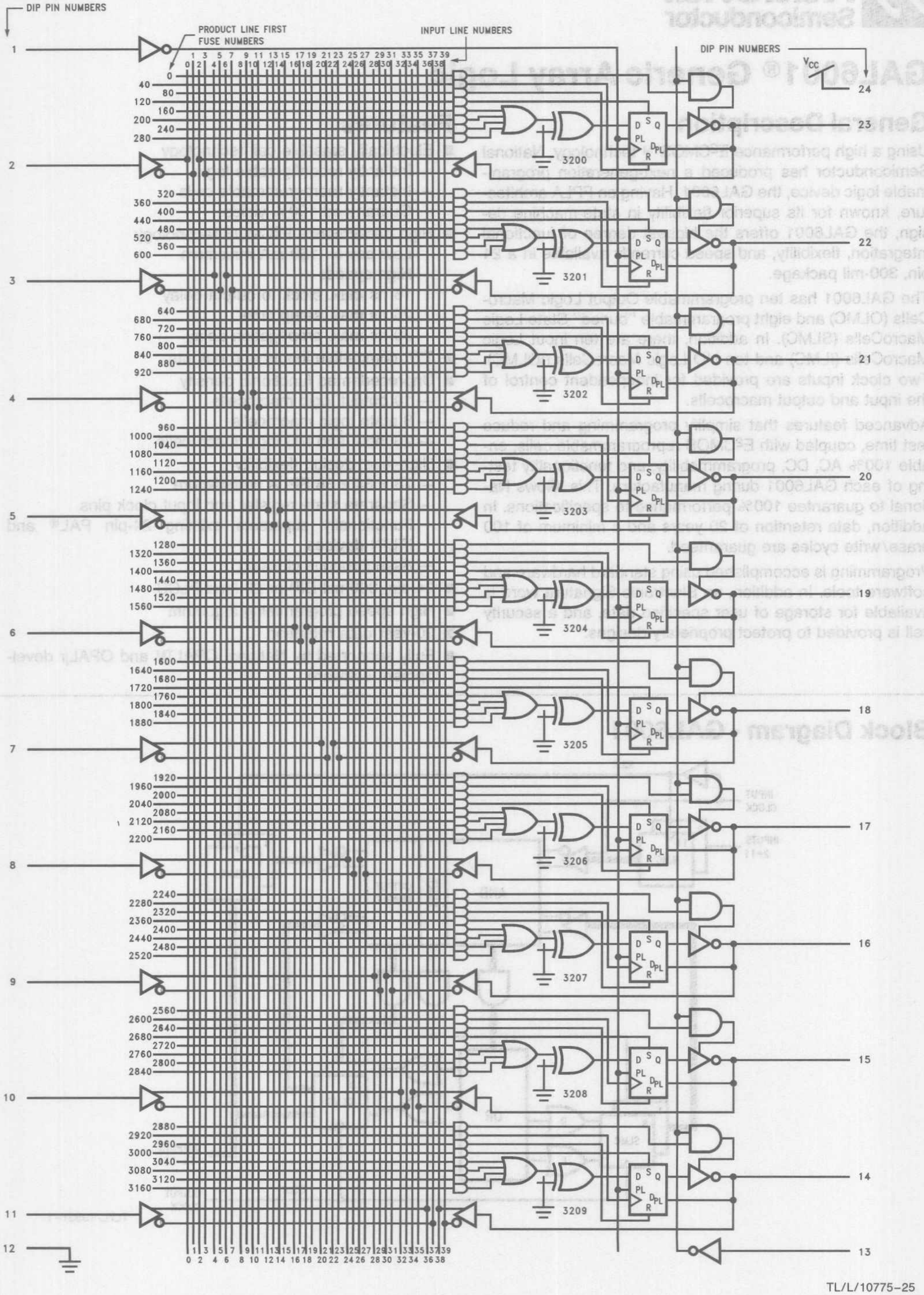
TL/L/10775-16



TL/L/10775-17

FIGURE 1. "RA" Output Logic Macrocell Logic Diagram

Logic Diagram—GAL20RA10



TL/L/10775-25

JEDEC Logic Array Cell Number = Product Line First Cell Number + Input Line Number.

GAL6001® Generic Array Logic

General Description

Using a high performance E2CMOS™ technology, National Semiconductor has produced a next-generation programmable logic device, the GAL6001. Having an FPLA architecture, known for its superior flexibility in state-machine design, the GAL6001 offers the highest degree of functional integration, flexibility, and speed currently available in a 24 pin, 300-mil package.

The GAL6001 has ten programmable Output Logic MacroCells (OLMC) and eight programmable "buried" State Logic MacroCells (SLMC). In addition, there are ten input Logic MacroCells (ILMC) and ten I/O Logic MacroCells (IOLMC). Two clock inputs are provided for independent control of the input and output macrocells.

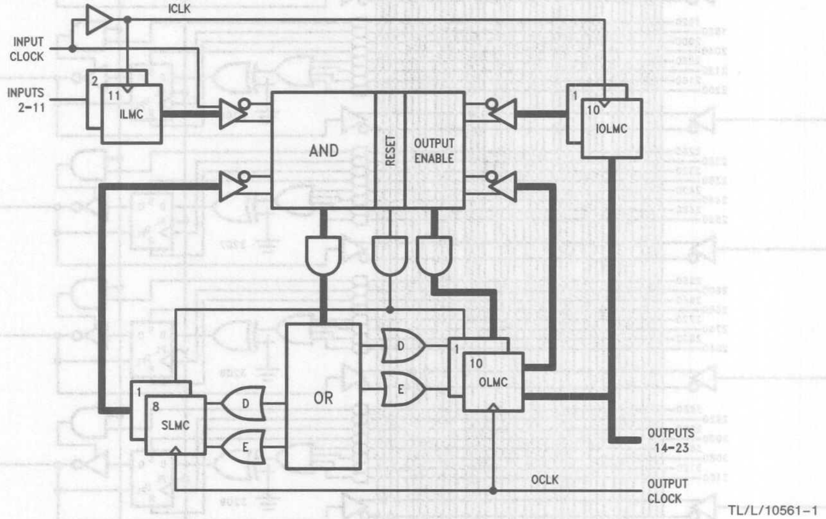
Advanced features that simplify programming and reduce test time, coupled with E2CMOS reprogrammable cells, enable 100% AC, DC, programmability, and functionality testing of each GAL6001 during manufacture. This allows National to guarantee 100% performance to specifications. In addition, data retention of 20 years and a minimum of 100 erase/write cycles are guaranteed.

Programming is accomplished using standard hardware and software tools. In addition, an Electronic Signature word is available for storage of user specified data, and a security cell is provided to protect proprietary designs.

Features

- Electrically erasable cell technology
  - — Instantly reconfigurable logic
  - — Instantly reprogrammable cells
  - — Guaranteed 100% yields
- High performance E2CMOS technology
  - — Low power: 150 mA maximum
  - — High speed:
    15 ns max. clock to output delay
    25 ns max. setup time
    30 ns max. propagation delay
- TTL compatible inputs and outputs
- Unprecedented functional density
  - — 10 output logic macrocells
  - — 8 state logic macrocells
  - — 20 input and I/O logic macrocells
- High-level design flexibility
  - — 78 x 64 x 36 FPLA architecture
  - — Separate state register and input clock pins
  - — Functionally supersets existing 24-pin PAL® and IFL™ devices
  - — Asynchronous clocking
- 24-pin, 300-mil DIP or 28-lead PLCC
- High speed programming algorithm
- 20-year data retention
- Fully supported by National OPAL™ and OPALjr development software

Block Diagram - GAL6001



TL/L/10561-1

Off-state Output Voltage Applied	-0.5 to V_{CC} $+1.0$V
Storage Temperature	0°C to +125°C

These are stress only ratings and functional operation of the device at these or at any other conditions above those indicated in the operational sections of this specification is not implied (while programming, follow the programming specifications).

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Temperature Range			Units
		Commercial			
		Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	V
T_A	Ambient Temperture	0		75	°C
T_C	Case Temperature	0		75	°C

Capacitance (Note 1) (T_A = 25°C, f = 1.0 MHz)

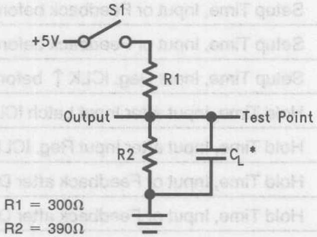
Symbol	Parameter	Test Conditions	Maximum*	Units
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V	8	pF
C_O	Output Capacitance	V_{CC} = 5.0V, V_O = 2.0V	10	pF
CB	Bidirectional Pin Cap	V_{CC} = 5.0V, V_B = 2.0V	10	pF

*Guaranteed but not 100% tested.

Switching Test Conditions

Input Pulse Levels	GND to 3.0V
Input Rise and Fall Times	5 ns (0.3V to 2.7V)
Input Timing Reference Levels	1.5V
Output Timing Reference Levels	1.5V
Output Load	See Figure

3-state levels are measured 0.5V from steady-state active level.



TL/L/10561-2

GAL6001 Reset Timing Specifications

Symbol	Parameter	Min	Typ	Max	Units
tPR	Reset Circuit Power-Up			100	ns
tRESET	Register Reset Time from Valid V_{CC}			45	μs

Electrical Characteristics over recommended operating conditions

Symbol	Parameter	Test Conditions		Temp Range	Min	Max	Units
I_{IH}, I_{IL}	Input Leakage Current	GND ≤ V_{IN} ≤ V_{CC} Max				±10	μA
I_{BZH}, I_{BZL}	Bidirectional Pin Leakage Current	GND ≤ V_{IN} ≤ V_{CC} Max				±10	μA
I_{CC}	Operating Power Supply Current	F = 15 MHz V_{CC} = V_{CC} Max		COM'L		150	mA
I_{OS}	Output Short Circuit (Note 1)	V_{CC} = 5.0V, V_{OUT} = GND			−30	−150	mA
V_{OL}	Output Low Voltage	V_{CC} = V_{CC} Min	I_{OL} = 16 mA	COM		0.5	V
V_{OH}	Output High Voltage	V_{CC} = V_{CC} Min	I_{OH} = −3.2 mA	COM	2.4		V
V_{IH}	Input High Voltage				2.0	V_{CC}+1	V
V_{IL}	Input Low Voltage					0.8	V

Note 1: One Output at a time for a maximum duration of one second.

Switching Characteristics over recommended operating conditions

Symbol	Description	Test Condition[1]	-30		Units
			Min	Max	
t_{pd1}	Combinatorial Input to Combinatorial Output	Note 1		30	ns
t_{pd2}	Feedback or I/O to Combinational Output	Note 1		30	ns
t_{coc}	Output D/E Reg. OCLK ↑ to Output Delay	Note 1		15	ns
t_{cot}	Output D Reg. Sum Term CLK ↑ to Output Delay	Note 1		35	ns
t_{su1}	Setup Time, Input before Input Latch ICLK ↓		2.5		ns
t_{su2}	Setup Time, Input before Input Reg. ICLK ↑		2.5		ns
t_{su3}	Setup Time, Input or Feedback before D/E Reg. OCLK ↑		25		ns
t_{su4}	Setup Time, Input or Feedback before D Reg. Sum Term CLK ↑		7.5		ns
t_{su5}	Setup Time, Input Reg. ICLK ↑ before D/E Reg. OCLK ↑		30		ns
t_{h1}	Hold Time, Input after Input Latch ICLK ↓		5		ns
t_{h2}	Hold Time, Input after Input Reg. ICLK ↑		5		ns
t_{h3}	Hold Time, Input or Feedback after D/E Reg. OCLK ↑		−5		ns
t_{h4}	Hold Time, Input or Feedback after D Reg. Sum Term CLK ↑		12.5		ns
f_{max}	Maximum Clock Frequency, OCLK			25	MHz
t_{en}	Input or I/O to Output Enabled	Note 2		25	ns
t_{dis}	Input or I/O to Output Disabled	Note 3		25	ns
t_{ar}	Input or I/O to Asynchronous Reg. Reset	Note 1		40	ns

Note 1: C_L = 50 pF, S1 closed

Note 2: Active High: S1 open, C_L = 50 pF
Active Low: S1 closed, C_L = 50 pF

Note 3: Active High: S1 open, C_L = 5 pF
Active Low: S1 closed, C_L = 5 pF

Differential Product Term (DPT) Switching Characteristics

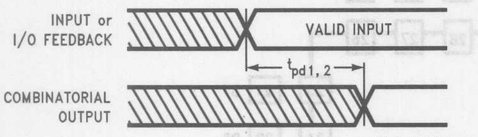
The number of DPT that may switch in the same direction at the same time is limited to a maximum of 15.

The number of DPT for a given design is calculated by subtracting the total number of Product-Terms that are switching from a Logical HI to a Logical LO from those switching from a Logical LO to a Logical HI within a 5 ns period.

$$DPT = (\text{P-Terms})_{LH} - (\text{P-Terms})_{HL}$$

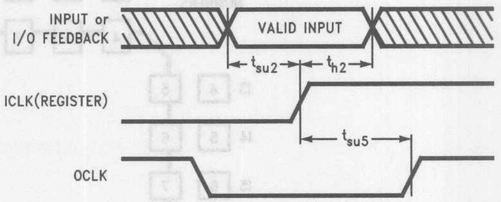
Switching Waveforms

Combinatorial Output



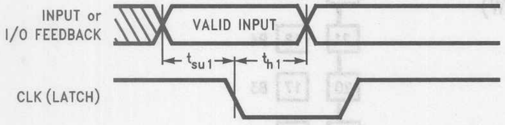
TL/L/10561-16

Registered Input



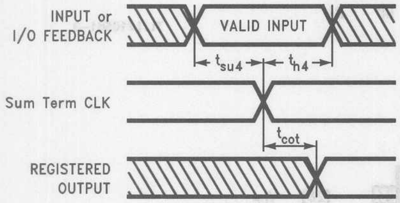
TL/L/10561-18

Latched Output



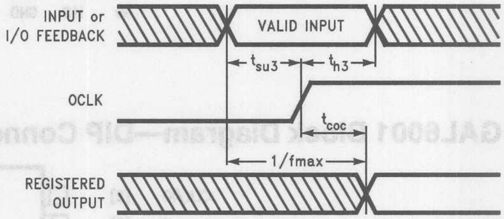
TL/L/10561-17

Registered Output (Sum Term CLK)



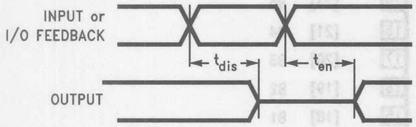
TL/L/10561-19

Registered Output



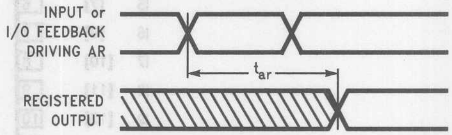
TL/L/10561-20

Input or I/O to Output Enable Disable



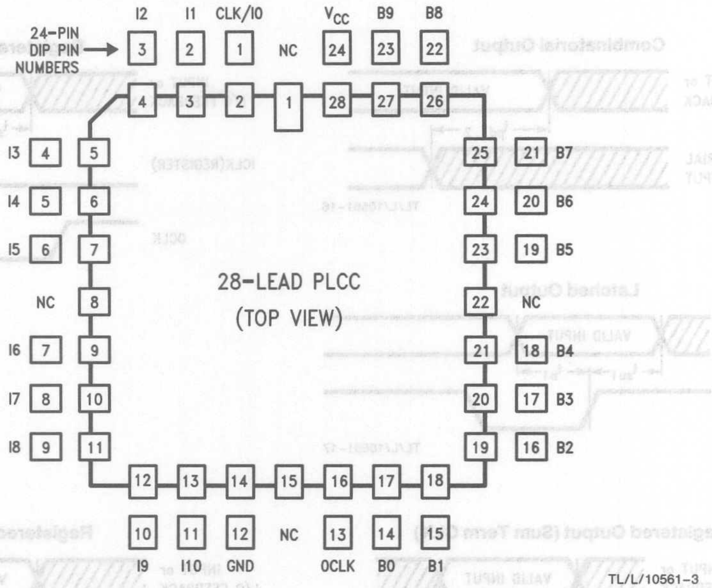
TL/L/10561-21

Asynchronous Reset



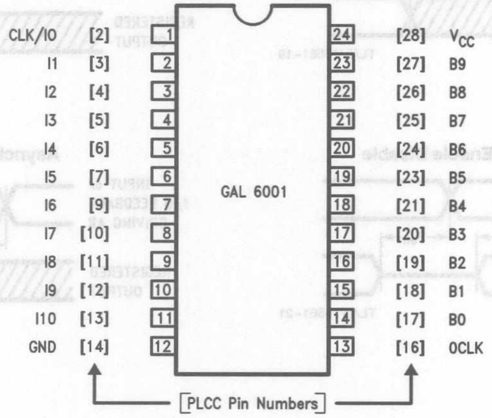
TL/L/10561-22

28-Lead PLCC Connection Diagram



TL/L/10561-3

GAL6001 Block Diagram—DIP Connections



TL/L/10561-4

Input Logic MacroCell (ILMC) and I/O Logic MacroCell (IOLMC)

The GAL6001 features two configurable input sections. The ILMC section corresponds to the dedicated input pins (2–11) and the IOLMC to the I/O pins (14–23). Each input section is configurable as a block for asynchronous, latched, or registered inputs. Pin 1 (ICLK) is used as an enable input for latched macrocells (transparent when high) and as a clock for registered macrocells (positive edge triggered).

Configurable input blocks can be used to advantage by system designers. Registered inputs are popular for synchronization and data merging. Transparent latches are useful when the input data is invalid outside a known time window. Direct inputs are used in systems where the input data is well ordered in time. With the GAL6001, external registers and latches are not necessary.

The various configurations of the input and I/O macrocells are controlled by programming four architecture control bits (INLATCH, INSYN, IOLATCH, and IOSYN) within the 68-bit architecture control word. The SYN bits determine whether the macrocells will have register/latch capability or will be strictly asynchronous. The LATCH bits select between latched and registered inputs.

The three valid macrocell configurations are shown in the macrocell equivalent diagrams on the following pages. The truth table associated with each diagram shows the values of the LATCH and SYN bits required to set the macrocell to the configuration shown.

Output Logic MacroCell (OLMC) and State Logic MacroCell (SLMC)

The outputs of the OR array feed two groups of macrocells. One group of eight macrocells is buried; its outputs feed back directly into the AND array rather than to device pins. These cells are called the State Logic MacroCells (SLMC), as they are useful for building state machines. The second group of macrocells consists of 10 cells whose outputs, in addition to feeding back into the AND array, are available at the device pins. Cells in this group are known as Output Logic MacroCells (OLMC).

Like the ILMC and IOLMC discussed above, output and state logic macrocells are configured by programming specific bits in the architecture control word (CKS(i), OUTSYN(i), XORD(i), XORE(i)), but unlike the input macrocells which must be configured in blocks, these macrocells are configurable on a macrocell-by-macrocell basis. Throughout this datasheet, i = [14 . . . 23] for OLMCs and i = [0 . . . 7] for SLMCs.

State and Output Logic MacroCells may be set to one of three valid configurations: combinational, D-type registered with sum term (asynchronous) clock, or D/E-type registered. Output macrocells always have I/O capability, with directional control provided by the 10 output enable (OE) product terms. Additionally, the polarity of each OLMC output is selectable through the XORD(i) architecture bits. Polarity selection is not necessary for SLMCs, since both the true and complemented forms of their outputs are available in the AND array. Polarity of all "E" sum terms is selectable through the XORE(i) architecture control bits.

When CKS(i) = 1 and OUTSYN(i) = 0, macrocell "i" is set as "D/E-type registered". In this configuration, the register is clocked from the common OCLK and the register clock enable input is controlled by the associated "E" sum term. This configuration is useful for building counters and state-machines with state hold functions.

When CKS(i) = 0 and OUTSYN(i) = 0, macrocell "i" is set as "D-type registered with sum term clock". In this configuration, the register is enabled and its "E" sum term is routed directly to the clock input. This allows for the popular "asynchronous programmable clock" feature, selectable on a register-by-register basis.

When CKS(i) = 0 and OUTSYN(i) = 1, macrocell "i" is set as "combinatorial". Configuring a SLMC in this manner turns it into a complement array. Complement arrays are used to construct multi-level logic.

Registers in both the Output and State Logic MacroCells feature a RESET input. This active high input allows the registers to be simultaneously and asynchronously reset from a common signal. The source of this signal is the RESET product term. Registers reset to a logic zero, but since the output buffers invert, a logic one will be present at the device pins.

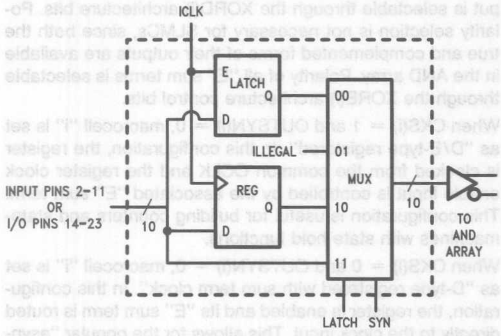
There are two possible feedback paths from each OLMC: one from before the output buffer (this is the normal path) and one from after the output buffer, through the IOLMCs. The second path is usable as a feedback only when the associated bi-directional pin is being used as an output; during input operations it becomes the input data path, turning the associated OLMC into an additional buried state macrocell.

The D/E registers used in this device offer the designer the ultimate in flexibility and utility. The D/E register construct can emulate RS-, JK-, and T-type registers with the same efficiency as a dedicated RS-, JK- or T-register.

The three valid macrocell configurations are shown in the macrocell equivalent diagrams on the following pages. The truth table associated with each diagram shows the bit value of CKS(i) and OUTSYN(i) required to set the macrocell to the configuration shown.

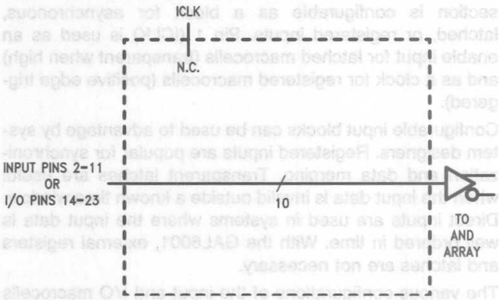
ILMC/IOLMC Configurations

ILMC/IOLMC Generic Block Diagram



TL/L/10561–5

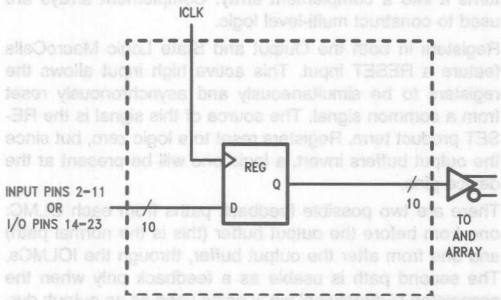
Asynchronous Input



TL/L/10561–6

LATCH	SYN
1	1

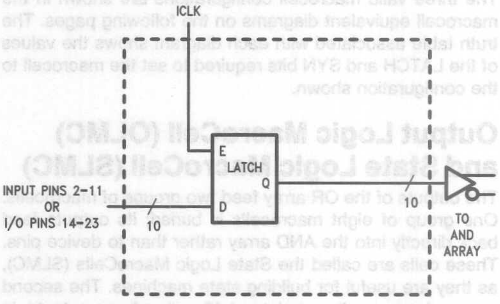
Registered Input



TL/L/10561–7

LATCH	SYN
1	0

Latched Input

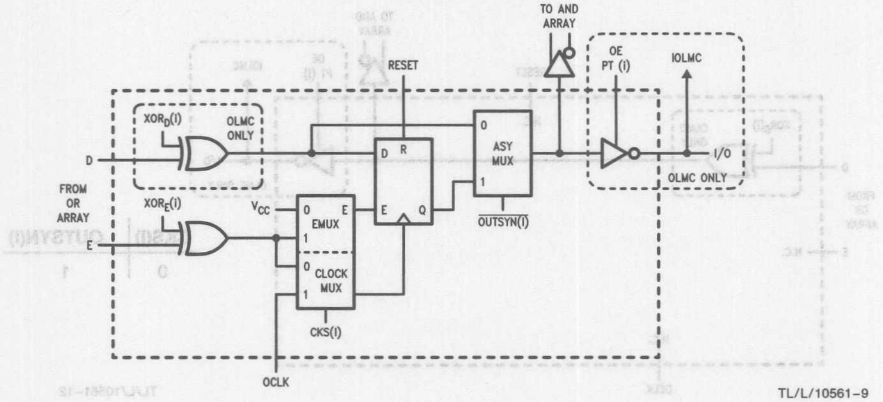


TL/L/10561–8

LATCH	SYN
0	0

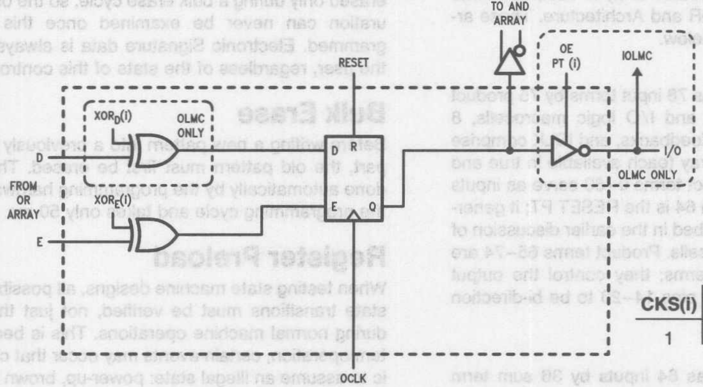
OLMC/SLMC Configurations

OLMC/SLMC Block Diagram



TL/L/10561–9

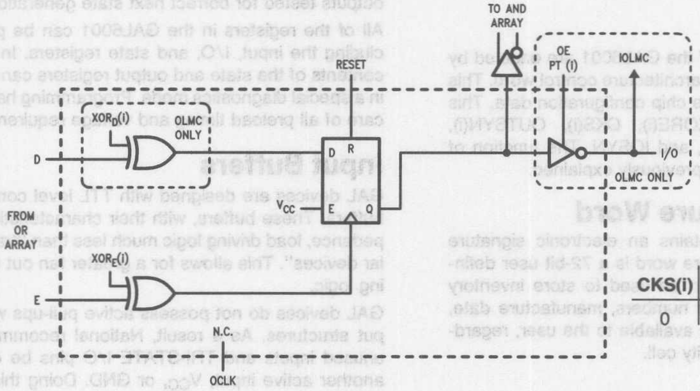
D/E Type Registered



CKS(i)	OUTSYN(i)
1	0

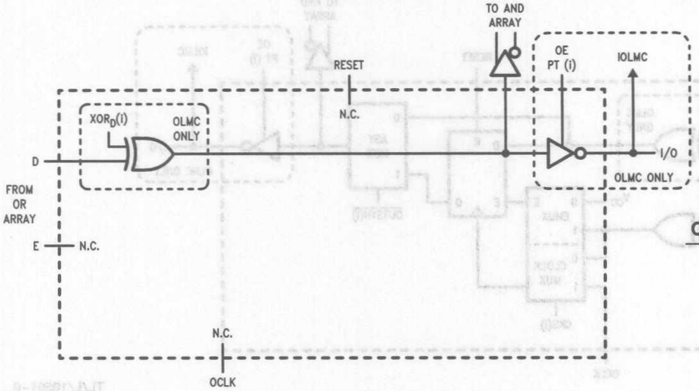
TL/L/10561–10

D-Type Registered with Sum Term Asynchronous Clock



CKS(i)	OUTSYN(i)
0	0

TL/L/10561–11



CKS(i)	OUTSYN(i)
0	1

TL/L/10561-12

Array Description

The GAL6001 E^2 reprogrammable array is subdivided into three smaller arrays: AND, OR and Architecture. These arrays are described in detail below.

AND ARRAY

The AND array is organized as 78 input terms by 75 product term outputs. The 20 input and I/O logic macrocells, 8 SLMC feedbacks, 10 OLMC feedbacks, and ICLK comprise a total of 39 inputs to this array (each available in true and complemented forms). Product terms 0–63 serve as inputs to the OR array. Product term 64 is the RESET PT; it generates the RESET signal described in the earlier discussion of output and state logic macrocells. Product terms 65–74 are the output enable product terms; they control the output buffers, thus enabling device pins 14–23 to be bi-direction or TRI-STATE®.

OR ARRAY

The OR array is organized as 64 inputs by 36 sum term outputs. Product terms 0–63 of the AND array serve as the inputs to this array. Of the 36 sum term outputs, 18 are data ("D") terms and 18 are enable/clock ("E") terms. These terms feed into the 10 OLMCs and 8 SLMCs, one "D" term and one "E" term to each.

ARCHITECTURE ARRAY

The various configurations of the GAL6001 are enabled by programming cells within the architecture control word. This 68-bit word contains all of the chip configuration data. This data includes: XORD(i), XORE(i), CKS(i), OUTSYN(i), INLATCH, INSYN, IOLATCH, and IOSYN. The function of each of these bits has been previously explained.

Electronic Signature Word

Every GAL6001 device contains an electronic signature word. The Electronic Signature word is a 72-bit user definable storage area, which can be used to store inventory control data, pattern revision numbers, manufacture date, etc. Signature data is always available to the user, regardless of the state of the security cell.

Security Cell

A security cell is provided with every GAL6001 device as a deterrent to unauthorized copying of the array patterns. Once programmed, this cell prevents further read access to the AND, OR and architecture arrays. This cell can be erased only during a bulk erase cycle, so the original configuration can never be examined once this cell is programmed. Electronic Signature data is always available to the user, regardless of the state of this control cell.

Bulk Erase

Before writing a new pattern into a previously programmed part, the old pattern must first be erased. This erasure is done automatically by the programming hardware as part of the programming cycle and takes only 50 ms.

Register Preload

When testing state machine designs, all possible states and state transitions must be verified, not just those required during normal machine operations. This is because in system operation, certain events may occur that cause the logic to assume an illegal state: power-up, brown out, line voltage glitches, etc. To test a design for proper treatment of these conditions, a method must be provided to break the feedback paths and force any desired state (i.e., illegal) into the registers. Then the machine can be sequenced and the outputs tested for correct next state generation.

All of the registers in the GAL6001 can be preloaded, including the input, I/O, and state registers. In addition, the contents of the state and output registers can be examined in a special diagnostics mode. Programming hardware takes care of all preload timing and voltage requirements.

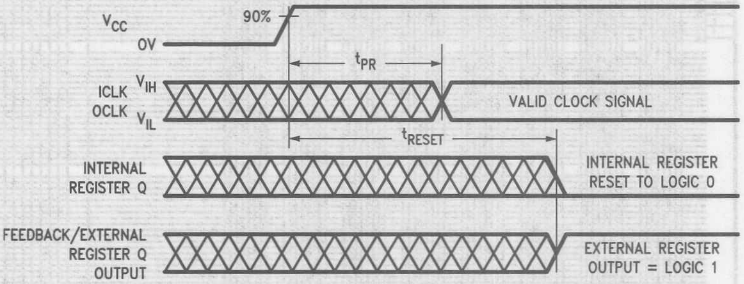
Input Buffers

GAL devices are designed with TTL level compatible input buffers. These buffers, with their characteristically high impedance, load driving logic much less than "traditional bipolar devices". This allows for a greater fan out from the driving logic.

GAL devices do not possess active pull-ups within their input structures. As a result, National recommends that all unused inputs and TRI-STATE I/O pins be connected to another active input, V_{CC}, or GND. Doing this will tend to improve noise immunity and reduce I_{CC} for the device.

result, the state on the registered output pins (if they are enabled) will always be high on power-up, regardless of the programmed polarity of the output pins. This feature can greatly simplify state machine design by providing a known state on power-up.

reset of the GAL6001. First, the V_{CC} rise must be monotonic. Second, the clock inputs must become a proper TTL level within the specified time (tPR). The registers will reset within a maximum of tRESET time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

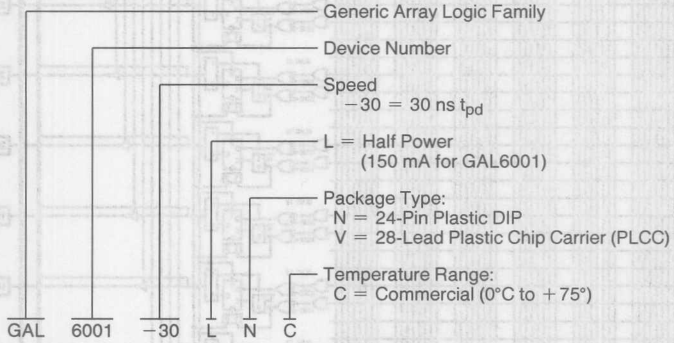


V_{CC}
90%
OV
t_{PR}
ICLK
OCLK
V_{IH}
V_{IL}
VALID CLOCK SIGNAL
t_{RESET}
INTERNAL REGISTER Q
INTERNAL REGISTER RESET TO LOGIC 0
FEEDBACK/EXTERNAL REGISTER Q OUTPUT
EXTERNAL REGISTER OUTPUT = LOGIC 1

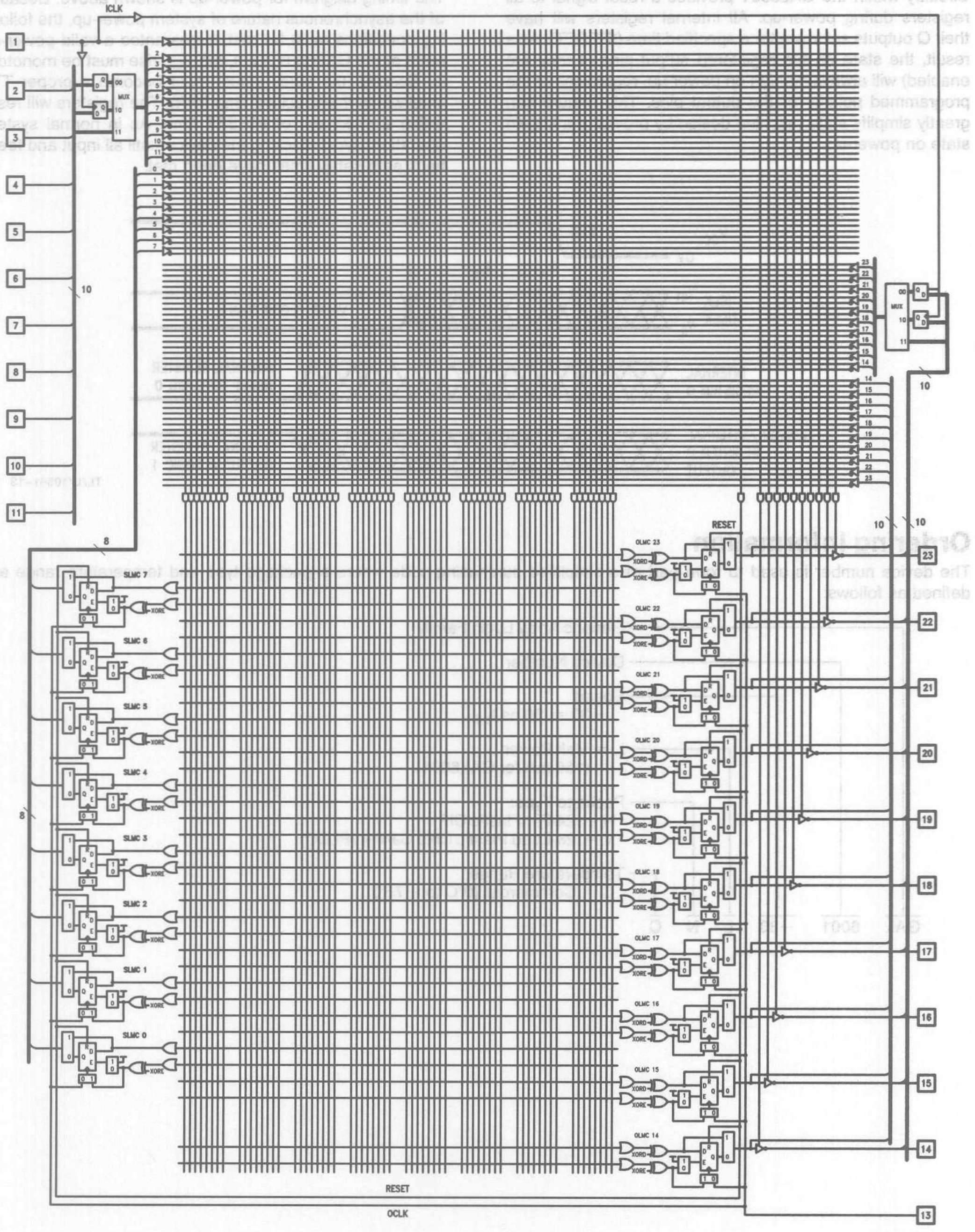
TL/L/10561–13

Ordering Information

The device number is used to form part of a simplified purchasing code where a package type and temperature range are defined as follows:



GAL6001 Logic Diagram



TL/L/10561–15

National Semiconductor

PAL10/10016P8
ECL Programmable Array Logic

General Description

The PAL1016P8/10016P8 is the first member of an ECL programmable logic device family possessing common electrical characteristics, utilizing an easily accommodated programming procedure, and produced with National Semiconductor's advanced oxide-isolated process. This family includes combinatorial, and registered output devices.

These devices are fabricated using National's proven Ti-W (Titanium-Tungsten) fuse technology to allow fast, efficient, and reliable programming.

This family allows the designer to quickly implement the defined logic function by removing the fuses required to properly configure the internal gates and/or registers. Product terms with all fuses removed assume a logical high state. All devices in this series are provided with an output polarity fuse that, if removed, will permit any output to independently provide a logic low when the equation is satisfied. When these fuses are intact the outputs provide a logic true (most positive voltage level) in response to the input conditions defined by the applicable equation. All input and I/O pins have on-chip 50 kΩ pull-down resistors.

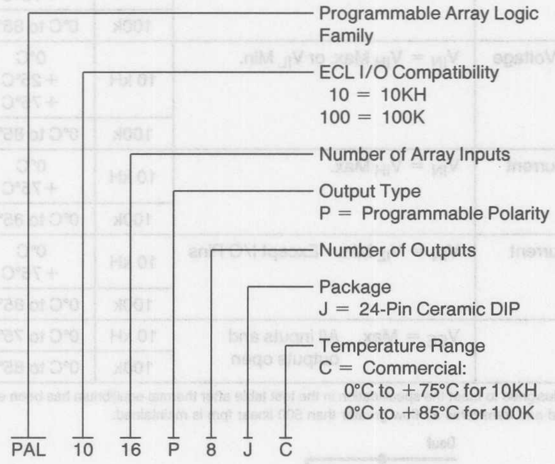
Fuse symbols have been omitted from the logic diagrams to allow the designer use of the diagrams to create fuse maps representing the programmed device.

All devices in this family can be programmed using conventional programmers. After the device has been programmed and verified, an additional fuse may be removed to inhibit further verification or programming. This "security" feature can provide a proprietary circuit which cannot easily be duplicated.

Features

- t_{PD} = 6 ns max
- Eight combinatorial outputs with programmable polarity
- Programmable replacement for conventional ECL logic
- Both 10KH and 100K I/O compatible versions
- Simplifies prototyping and board layout
- 24-pin thin DIP packages.
- Programmed on conventional TTL PLD programmers
- Security fuse to prevent direct copying
- Reliable titanium-tungsten fuses

Ordering Information



Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias (Ambient)	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
V_{EE} Relative to V_{CC}	−7V to +0.5V
Any Input Relative to V_{CC}	V_{EE} to +0.5V
Lead Temperature (Soldering, 10 seconds)	300°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500 Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028

Recommended Operating Conditions

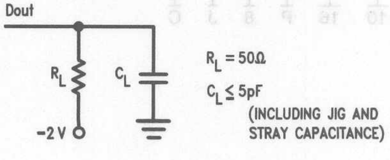
Symbol	Parameter		Min	Typ	Max	Units
V_{EE}	Supply Voltage	10 kH	−5.46	−5.2	−4.94	V
		100k	−4.73	−4.5	−4.27	
R_L	Standard 10 kH/100k Load			50		Ω
C_L	Standard 10 kH/100k Load			5		pF
T_A	Operating Ambient Temperature	10 kH	0		+75	°C
		100k	0		+85	

Electrical Characteristics

Over Recommended Operating Conditions. Output Load = 50Ω to −2.0V.

Symbol	Parameter	Conditions		T_A	Min	Max	Units
V_{IH}	High Level Input Voltage	Guaranteed input voltage high for all inputs	10 kH	0°C +25°C +75°C	−1170 −1130 −1070		mV
			100k	0°C to 85°C	−1165	−880	
V_{IL}	Low Level Input Voltage	Guaranteed input voltage low for all inputs	10 kH	0°C +25°C +75°C		−1480 −1480 −1450	mV
			100k	0°C to 85°C	−1810	−1475	
V_{OH}	High Level Output Voltage	V_{IN} = V_{IH} Max. or V_{IL} Min.	10 kH	0°C +25°C +75°C	−1020 −980 −920	−840 −810 −735	mV
			100k	0°C to 85°C	−1025	−880	
V_{OL}	Low Level Output Voltage	V_{IN} = V_{IH} Max. or V_{IL} Min.	10 kH	0°C +25°C +75°C	−1950 −1950 −1950	−1630 −1630 −1600	mV
			100k	0°C to 85°C	−1810	−1620	
I_{IH}	High Level Input Current	V_{IN} = V_{IH} Max.	10 kH	0°C +75°C		220	μA
			100k	0°C to 85°C			
I_{IL}	Low Level Input Current	V_{IN} = V_{IL} Min. Except I/O Pins	10 kH	0°C +75°C	0.5		μA
			100k	0°C to 85°C			
I_{EE}	Supply Current	V_{EE} = Max. All inputs and outputs open	10 kH	0°C to 75°C	−240		mA
			100k	0°C to 85°C			

Note: This product family has been designed to meet the specification in the test table after thermal equilibrium has been established. The circuit is in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained.



TL/L/6161-4

Switching Characteristics

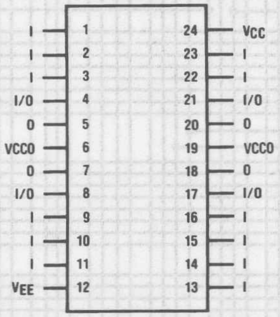
Over Recommended Operating Conditions; Output load: $R_L = 50\Omega$ to $-2.0V$, $C_L = 5$ pF to GND.

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
t_{PD}	Input to Output*			4	6	ns
t_r	Output Rise Time		0.5	1	2.5	ns
t_f	Output Fall Time		0.5	1	2.5	ns

*Measure t_{PD} at threshold points

Connection Diagram

Dual-In-Line Package



TL/L/6161-2

Top View

PAL Design

The first step in designing a PAL device is the selection of the appropriate device to accommodate the logic equations. This is accomplished by partitioning the system into logic blocks with a defined number of inputs and outputs. Next, a device with an equal or greater I/O capability is selected to implement each logic block. The assignment of inputs and outputs to specific pins follows the device selection.

This device selection procedure is most easily accomplished with the use of computer software such as the OPAL™ package of programs by National Semiconductor Corporation, but can be done manually using the logic diagram and logic symbols provided in this document.

Specifying the Fuse Pattern

Once a device with pinout is selected, the fuse pattern may be specified. The best procedure is the use of the PLAN, or a similar software package which will create the fuse pattern from the defined logic for the device and download the pattern to a programmer. Most common device programmers are provided with an RS-232 port which accesses the data provided in JEDEC or a selected HEX format.

Logic diagrams can be translated to PAL logic diagrams if desired. Fuses left intact are indicated on the logic diagram by an "X" at the intersection of the input line and the AND gate product line. A blown fuse is not marked. The PAL logic diagrams are provided with no fuse locations marked, allowing the designer to use the diagram to manually create a fuse map. Actually, the unprogrammed device is shipped with all Xs (fuses) intact. Each fuse node is identified by a product line number and an input line number.

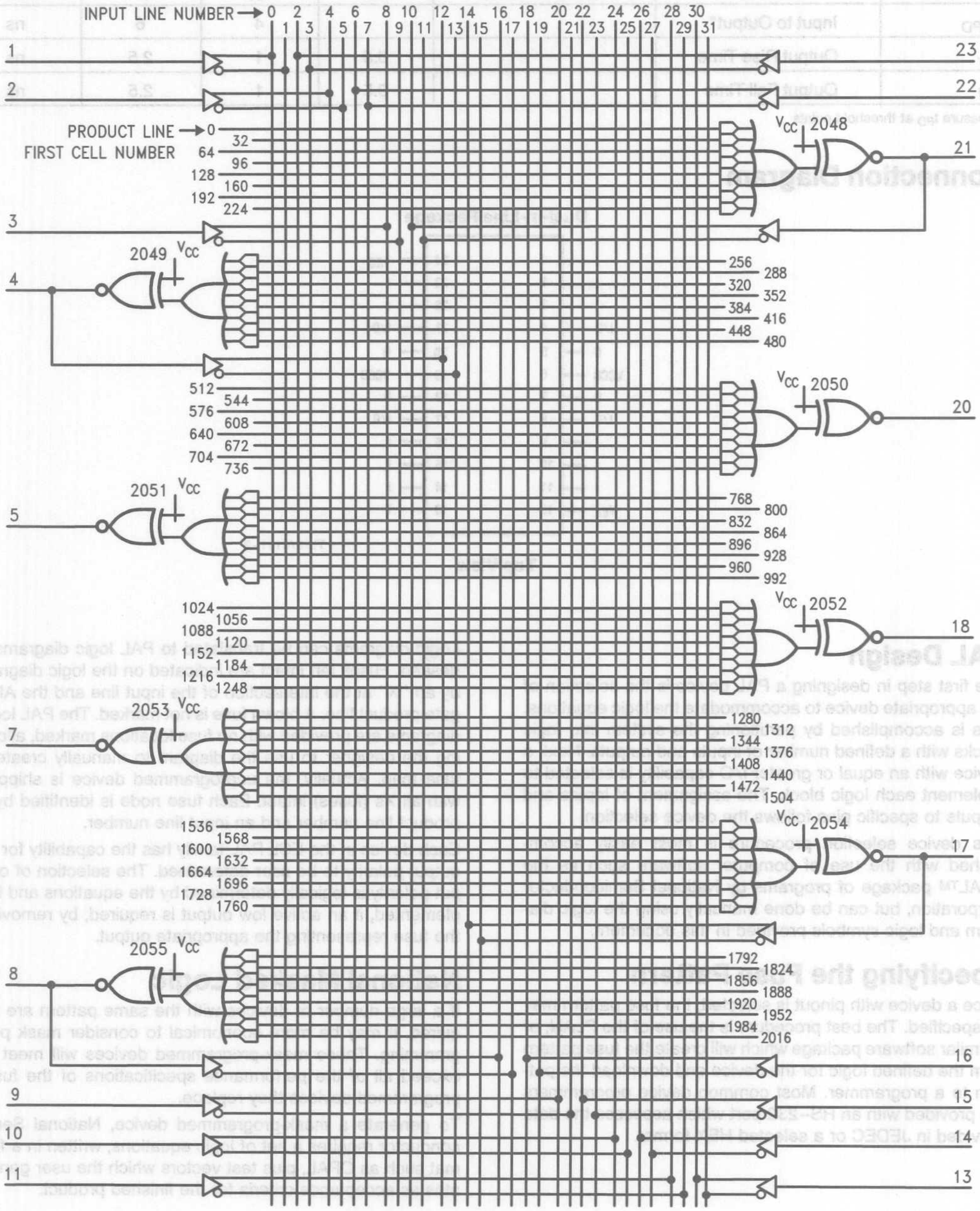
Each device in the ECL PAL family has the capability for its output polarity to be user-determined. The selection of output polarity is logically determined by the equations and implemented, if an active low output is required, by removing the fuse representing the appropriate output.

National Masked Logic

If a large number of devices with the same pattern are required, it may be more economical to consider mask programming. These mask-programmed devices will meet or exceed all of the performance specifications of the fuse-programmed devices they replace.

To generate a mask-programmed device, National Semiconductor requires a set of logic equations, written in a format such as OPAL, plus test vectors which the user generates as acceptance criteria for the finished product.

Logic Diagram PAL1016P8/PAL10016P8



TL/L/6161-3

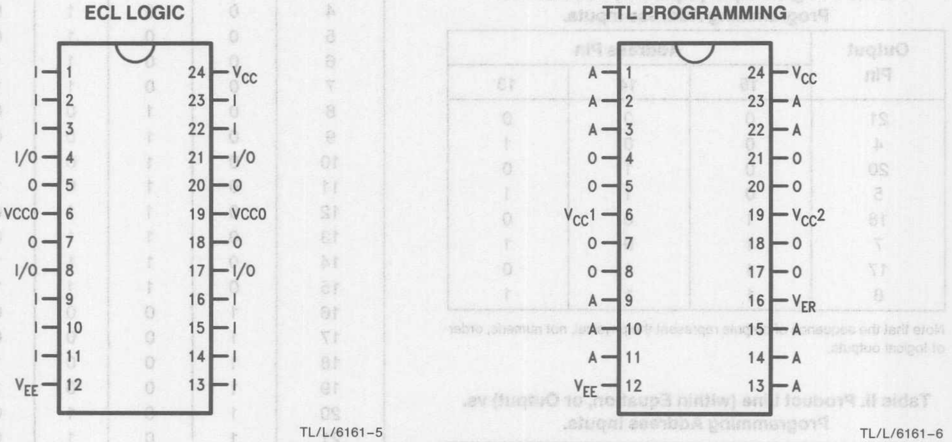
JEDEC logic array cell number = product line first cell number + input line number.

Programming Specification

This specification defines the programming and verification procedure for the first programmable logic devices in Nationals generic ECL family. The internal fuse arrays consists of 64 product lines (8 for each output), each containing 32 fuse locations (1 for each of 16 inputs and its complement) for a total of 2048 array fuses. Eight additional fuses exist to allow changing the active output polarity.

Each ECL device is programmed and verified as a 2048x1 TTL PROM. The connection diagrams in *Figure 1* illustrate the difference between the logical ECL device and the PROGRAMMABLE TTL device.

For a list of current software and programming support tools available for these devices, please contact your local National Semiconductor sales representative or distributor.



TL/L/6161–5

TL/L/6161–6

FIGURE 1. Connection Diagrams

Array Fuse Addressing

When programming or verifying a fuse location, the output (equation) is addressed by the 3 address pins 13, 14, and 15. The eight product lines, within the equation, are selected by the 3 address pins 9, 10, and 11. The fuse pair locations representing the logical inputs are selected by the 4 address pins 2, 3, 22, and 23, with the complementing fuse within the pair by the address pin 1. The programming address data is detailed in Tables I–III.

Table I. Logic Output (Equation) Selection vs. Programming Address Inputs.

Output Pin	Address Pin		
	15	14	13
21	0	0	0
4	0	0	1
20	0	1	0
5	0	1	1
18	1	0	0
7	1	0	1
17	1	1	0
8	1	1	1

Note that the sequence of outputs represent the physical, not numeric, order of logical outputs.

Table II. Product Line (within Equation, or Output) vs. Programming Address Inputs.

Product Pin	Address Pin		
	11	10	9
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table III. Input Line Selection vs. Programming Address Inputs.

Input Line	Address Pin				
	23	22	3	2	1
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1

Note pin 1 affects complementing fuse only.

Fuse Programming and Verification

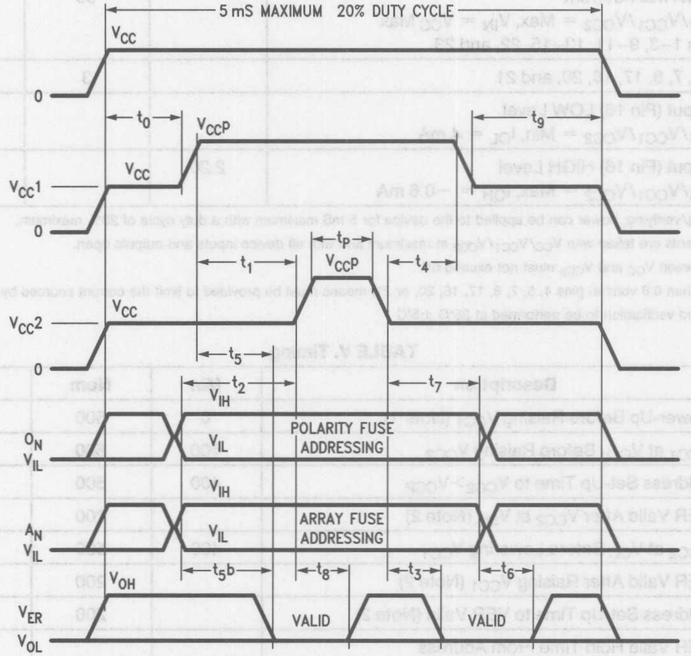
The array and output polarity fuse programming waveform diagram is shown in *Figure 2*. The 8 output pins 0_N are used only to change the polarity of the selected device output and for removing the "security" fuse. Tables 4 and 5 define the voltage and timing requirements.

Programming Procedure

1. Power is applied to the device. VCC, VCC1, and VCC2 (pins 24, 6, and 19) go to VCC. (The voltage applied to pin 24 cannot precede the voltage applied to pin 6) The output pins (4, 5, 7, 8, 17, 18, 20, and 21), are open circuited, or held at a logic low level, while programming the array.
2. After T0, VCC1 (pin 6) can be raised from 5.0 to 10.75V at a slew rate not to exceed 10V/μS, or not less than 1V/μs.
3. The 11 address inputs (pins 1-3, 9-11, 13-15, 22, and 23) will define the location of the array fuse to be opened or the applicable output pin will define the polarity fuse to be opened.
4. After VCC1 has been stable at 10.75V for period T1 and the address has been stable defining the applicable fuse location for period T2, VCC2 (pin 19) may slew from 5.0 to 10.75V at a slew rate of 1 to 10V/μS.
5. VCC2 must remain stable at 10.75V for the duration of the programming pulse (TP) before returning to 5.0V.
6. With VCC1 at 10.75V and after VCC2 has been stable at 5.0V for the period T3, VER pin (16) may be sampled. If the fuse was properly opened, a logic low level will be observed. If the fuse did not open, steps 4 through 6 may be repeated up to 15 times.
7. If additional locations are to be addressed, steps 3 through 6 must be repeated for each fuse to be opened while observing the maximum power up time and duty cycle.

Fuse Verification

Fuse verification may be performed independent of programming. As seen in *Figure 2*, with VCC1 at VCCP and VCC2 at VCC verification may occur within the defined timing constraints. (See Table V)



TL/L/6161-7

FIGURE 2. Array/Polarity Programming Diagram

TABLE IV. DC Requirements

Symbol	Description	Min	Nom	Max	Units
V_{CC}	Pin 24 Voltage While Programming or Verifying (Pin 19 Verifying) (Note 1)	4.75	5.00	5.25	V
I_{CC}	Pin 24 Current While Programming (Note 2)		200	300	mA
V_{CCp}	V_{CC1}/V_{CC2} (Pins 6 and 19) Voltage While Programming (Note 3)	10.50	10.75	11.00	V
I_{CC1}	V_{CC1} (Pin 6) Current While Programming (Note 2)		300	450	mA
I_{CC2}	VCC2 (Pin 19) Current While Programming (Note 2)		10	25	mA
VIL	Input LOW Level - If Left Open, Pins 4, 5, 7, 8, 17, 18, 20, and 21 are Held Low by Internal 50K Resistor	0		0.8	V
I_{IL}	Input LOW Current - Pins; 1–3, 9–11, 13–15, 22, and 23 $V_{CC}/V_{CC1}/V_{CC2}$ = Max, V_{IN} = 0.4V		−1.0	−1.5	mA
	4, 5, 7, 8, 17, 18, 20, and 21 (Note 4) $V_{CC}/V_{CC1}/V_{CC2}$ = Max, V_{IN} = 0.8V		−0.25	−1.5	mA
V_{IH}	Input HIGH Level	2.20		V_{CC}	V
I_{IH}	Input HIGH Current $V_{CC}/V_{CC1}/V_{CC2}$ = Max, V_{IN} = V_{CC} Max Pins 1–3, 9–11, 13–15, 22, and 23		90	300	µA
	4, 5, 7, 8, 17, 18, 20, and 21		3	5	mA
V_{OL}	Output (Pin 16) LOW Level $V_{CC}/V_{CC1}/V_{CC2}$ = Min, I_{OL} = 4 mA			0.8	V
V_{OH}	Output (Pin 16) HIGH Level $V_{CC}/V_{CC1}/V_{CC2}$ = Max, I_{OH} = −0.6 mA	2.20			V

Note 1: While programming/verifying, power can be applied to the device for 5 mS maximum with a duty cycle of 20% maximum.

Note 2: Current measurements are taken with $V_{CC}/V_{CC1}/V_{CC2}$ at maximum and with all device inputs and outputs open.

Note 3: The difference between V_{CC} and V_{CCp} must not exceed 6V.

Note 4: If V_{IN} (V_{IL}) is less than 0.8 volts at pins 4, 5, 7, 8, 17, 18, 20, or 21, means must be provided to limit the current sourced by the device pins to 10 mA.

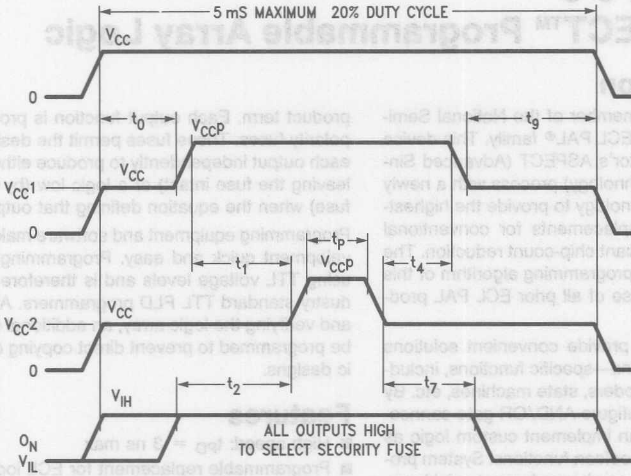
Note 5: All programming and verification to be performed at 25°C ±5°C

TABLE V. Timing

Symbol	Description	Min	Nom	Max	Units
T0	Power-Up Before Raising V_{CC1} (Note 1)	0	500		ns
T1	V_{CC1} at V_{CCp} Before Raising V_{CC2}	400	500		ns
T2	Address Set-Up Time to $V_{CC2}>V_{CCP}$	400	500		ns
T3	VER Valid After V_{CC2} at V_{CC} (Note 2)		200	500	ns
T4	V_{CC2} at V_{CC} Before Lowering V_{CC1}	400	500		ns
T5	VER Valid After Raising V_{CC1} (Note 2)		200	500	ns
T5b	Address Set-Up Time to VER Valid (Note 2)		200	500	ns
T6	VER Valid Hold Time From Address			0	ns
T7	V_{CC2} at V_{CC} Before Address Change	400	500		ns
T8	VER Valid Hold Time From $V_{CC2} > V_{CCP}$ (Note 2)	0	100		ns
T9	V_{CC1} at V_{CC} Before Power Down	0			ns
TP	Programming Pulse	10	10	30	µs

Note 1: Observe the maximum power-up time or 5 ms and duty cycle of 20% for $V_{CC}/V_{CC1}/V_{CC2}$ during programming.

Note 2: VER is valid when V_{CC2} = V_{CC} and V_{CC1} = V_{CCP}.



TL/L/6161–8

FIGURE 3. Security Fuse Programming Diagram

National Semiconductor

PAL10/10016P8-3
3 ns ECL ASPECT™ Programmable Array Logic

General Description

The PAL10/10016P8-3 is a member of the National Semiconductor 28-pin high speed ECL PAL® family. This device utilizes National Semiconductor's ASPECT (Advanced Single Poly Emitter Coupled Technology) process with a newly developed tungsten fuse technology to provide the highest-speed user-programmable replacements for conventional ECL SSI-MSI logic with significant chip-count reduction. The JEDEC fuse-map format and programming algorithm of this device is compatible with those of all prior ECL PAL products from National.

Programmable logic devices provide convenient solutions for a wide variety of applications—specific functions, including random logic, custom decoders, state machines, etc. By programming fuse links to configure AND/OR gate connections, the system designer can implement custom logic as convenient sum-of-products Boolean functions. System prototyping and design iterations can be performed quickly using these off-the shelf products.

The PAL10/10016P8-3 logic array has a total of 16 complementary input pairs, 64 product terms and 8 programmable polarity output functions. Each output function is the OR-sum of 8 product terms. Each product term is satisfied when all array inputs which are connected to it (via intact fuses) are in the correct state as defined by the equation for that product term. Each output function is provided with output polarity fuses. These fuses permit the designer to configure each output independently to produce either a logic high (by leaving the fuse intact) or a logic low (by programming the fuse) when the equation defining that output is satisfied.

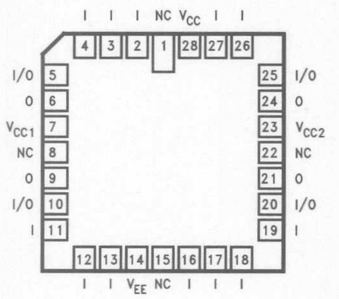
Programming equipment and software make PAL design development quick and easy. Programming is accomplished using TTL voltage levels and is therefore supported by industry standard TTL PLD programmers. After programming and verifying the logic array, an additional security fuse may be programmed to prevent direct copying of proprietary logic designs.

Features

- High speed: t_{PD} = 3 ns max
- Programmable replacement for ECL logic
- Both 100K and 10 KH I/O compatible versions
- Eight output functions with programmable polarity
- Improved programmability tungsten fuses
- Security fuse to prevent direct copying
- Programmed on conventional TTL PLD programmers
- Fully supported by OPAL™ and OPALjr development software
- Commercial and Military ranges

Connection Diagram

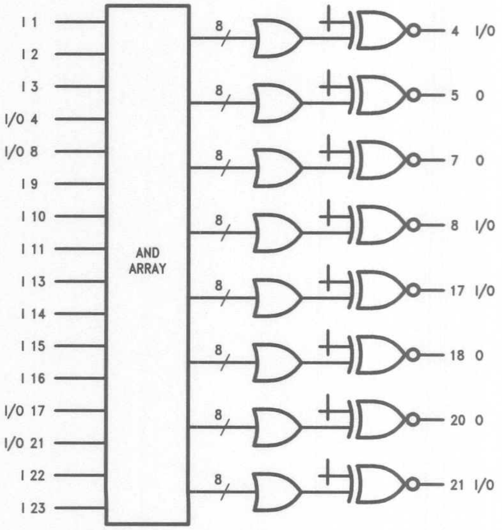
PLCC Pin Out



TL/L/10714–6

Block Diagram

PAL10/10016P8-3



TL/L/10714–1

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature under Bias	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
V_{EE} Relative to V_{CC}	−7V to +0.5V
Input Voltage	V_{EE} to +0.5V

Output Current	−50 mA
Lead Temperature (Soldering, 10 Seconds)	300°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028

Recommended Operating Conditions for Commercial Range

Symbol	Parameter		Min	Typ	Max	Units
V_{EE}	Supply Voltage	10KH	−5.46	−5.2	−4.94	V
		100K	−4.80	−4.5	−4.20	
T	Operating Temperature (Note)	10KH	0		+75	°C
		100K	0		+85	

Electrical Characteristics Over Recommended Operating Conditions Output Load = 50Ω to −2.0V

Symbol	Parameter	Conditions	T_A		Min	Max	Units
V_{IH}	High Level Input Voltage	Guaranteed Input Voltage High for All Inputs	10KH	0°C	−1170	−840	mV
				+25°C	−1130	−810	
				+75°C	−1070	−735	
			100K	0°C to +85°C	−1165	−880	
V_{IL}	Low Level Input Voltage	Guaranteed Input Voltage Low for All Inputs	10KH	0°C	−1950	−1480	mV
				+25°C	−1950	−1480	
				+75°C	−1950	−1450	
			100K	0°C to +85°C	−1810	−1475	
V_{OH}	High Level Output Voltage	V_{IN} = V_{IH} Max or V_{IL} Min	10KH	0°C	−1020	−840	mV
				+25°C	−980	−810	
				+75°C	−920	−735	
			100K	0°C to +85°C	−1025	−880	
V_{OL}	Low Level Output Voltage	V_{IN} = V_{IH} Max or V_{IL} Min	10KH	0°C	−1950	−1630	mV
				+25°C	−1950	−1630	
				+75°C	−1950	−1600	
			100K	0°C to +85°C	−1810	−1620	
I_{IH}	High Level Input Current	V_{IN} = V_{IH} Max	10KH	0°C +75°C		220	μA
			100K	0°C to +85°C			
I_{IL}	Low Level Input Current	V_{IN} = V_{IL} Min	10KH	0°C +75°C	0.5		μA
			100K	0°C to +85°C			
I_{EE}	Supply Current	V_{EE} = Min All Inputs and Outputs Open	10KH	0°C to +75°C	−230		mA
			100K	0°C to +85°C			

Note: Operating temperatures for circuits in J and N packages are specified as ambient temperatures (T_A) with circuits in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained.

Switching Characteristics

Over Recommended Operating Conditions, Output load: $R_L = 50\Omega$ to $-2.0V$, $C_L = 5$ pF to GND

Symbol	Parameter	Measured Test Conditions	Commercial		Units
			Min	Max	
t_{PD}	Input to Output	Measured at Threshold Points (Note 1)		3.0	ns
t_r	Output Rise Time	Measured between 20% and 60% Points	0.25	1.25	ns
t_f	Output Fall Time		0.25	1.25	ns

Note 1: All AC Measurements are to be made from Threshold Point.

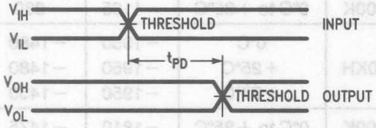
V_{IH} = Threshold + 400 mV

V_{IL} = Threshold − 400 mV

$$\text{Threshold} = \frac{V_{IH_{Min}} + V_{IL_{Max}}}{2}$$

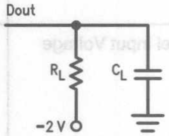
Part	Temp	$V_{IN_{Min}}$	$V_{IL_{Max}}$	Threshold	V_{IH}	V_{IL}
10 kH	−55°C	−1250	−1480	−1365	−965	−1765
	0°C	−1170	−1480	−1325	−925	−1725
	25°C	−1130	−1480	−1300	−900	−1700
	75°C	−1070	−1450	−1260	−860	−1660
	125°C	−1000	−1420	−1210	−810	−1610
100k	All	−1165	−1475	−1300	−900	−1700

Timing Measurements



TL/L/10714-2

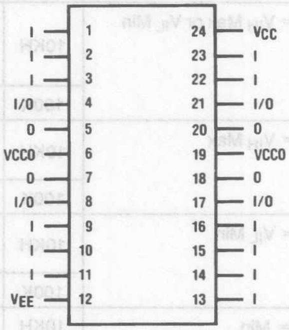
Test Load



TL/L/10714-3

Connection Diagram

Dual-In-Line Package



TL/L/10714-4

Top View

Functional Testing

As with all field-programmable devices, the user of the ECL PAL devices provides the final manufacturing step. While National's PAL devices undergo extensive testing when they are manufactured, their logic function can be fully tested only after they have been programmed to the user's pattern.

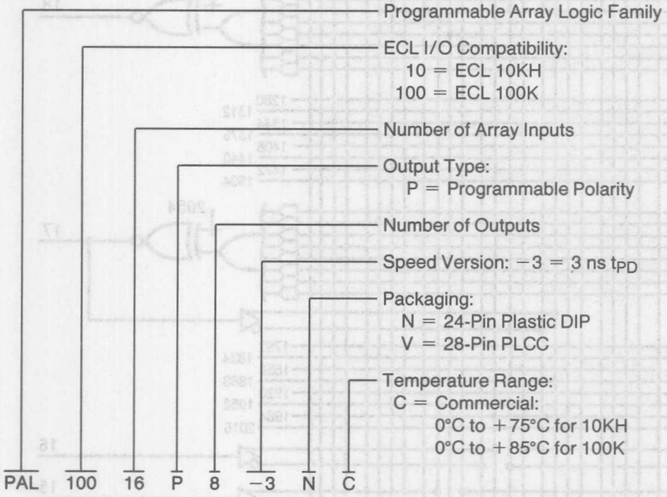
To ensure that the programmed PAL devices will operate properly in your system, National Semiconductor (along with most other manufacturers of PAL devices) strongly recommends that devices be functionally tested before being installed in your system. Even though the number of post-programming functional failures is small, testing the logic function of the PAL devices before they reach system assembly will save board debugging and rework costs. For more information about the functional testing of PAL devices, please refer to National Semiconductor's Application Note #351 and the *Programmable Logic Design Guide*.

Design Development Support

A variety of software tools and programming hardware is available to support the development of designs using PAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate JEDEC-compatible "fuse maps". The industry-standard JEDEC format ensures that the resulting fuse-map files can be downloaded into a large variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL™ software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

A detailed logic diagram showing all JEDEC fuse-map addresses for the PAL10/10016P8-3 is provided for direct map editing and diagnostic purposes. For a list of current software and programming support tools available for these devices, please contact your local National Semiconductor sales representative or distributor. If detailed specifications of the ECL PAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support Department.

Ordering Information



Programmable Array Logic Family

ECL I/O Compatibility:
10 = ECL 10KH
100 = ECL 100K

Number of Array Inputs

Output Type:
P = Programmable Polarity

Number of Outputs

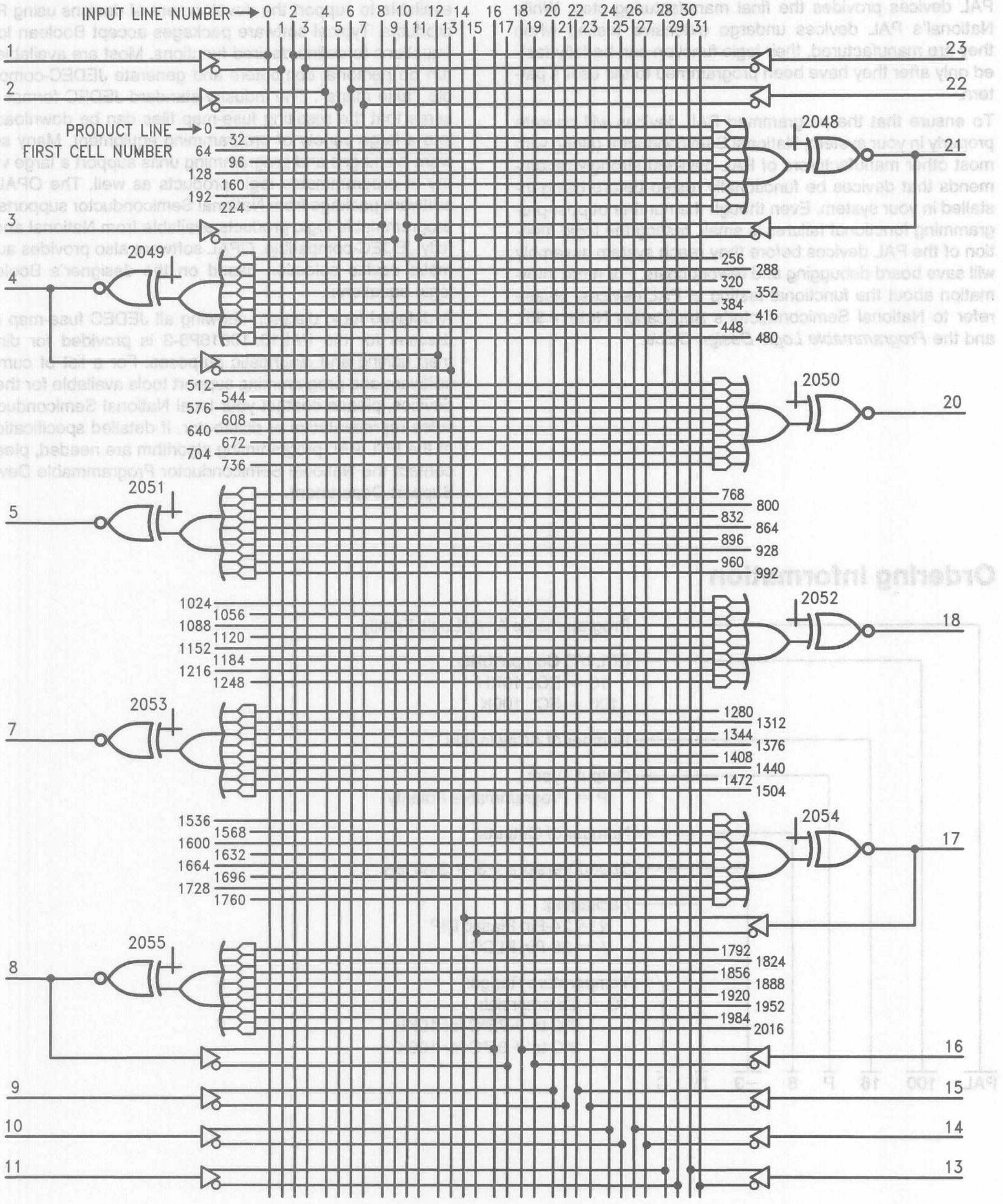
Speed Version: −3 = 3 ns t_{PD}

Packaging:
N = 24-Pin Plastic DIP
V = 28-Pin PLCC

Temperature Range:
C = Commercial:
0°C to +75°C for 10KH
0°C to +85°C for 100K

PAL 100 16 P 8 −3 N C

Logic Diagram—PAL1016P8-3/PAL10016P8-3



TL/L/10714–5

National Semiconductor

PAL10/10016PE8-3 (PLCC Only) 3 ns ECL ASPECT™ Programmable Array Logic

General Description

The PAL10/10016PE8-3 is a member of the National Semiconductor 28-pin high speed ECL PAL® family. This device utilizes National Semiconductor's ASPECT (Advanced Single Poly Emitter Coupled Technology) process with a newly developed tungsten fuse technology to provide the highest-speed user-programmable replacements for conventional ECL SSI-MSI logic with significant chip-count reduction. The JEDEC fuse-map format and programming algorithm of this device is compatible with those of all prior ECL PAL products from National.

Programmable logic devices provide convenient solutions for a wide variety of applications—specific functions, including random logic, custom decoders, state machines, etc. By programming fuse links to configure AND/OR gate connections, the system designer can implement custom logic as convenient sum-of-products Boolean functions. System prototyping and design iterations can be performed quickly using these off-the shelf products.

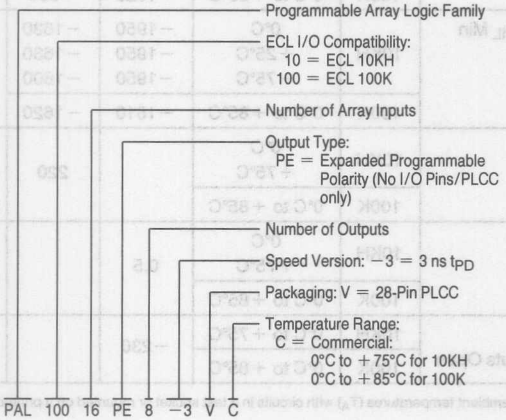
The PAL10/10016PE8-3 logic array has a total of 16 complementary input pairs, 64 product terms and 8 programmable polarity output functions. Each output function is the OR-sum of 8 product terms. Each product term is satisifed when all array inputs which are connected to it (via intact fuses) are in the correct state as defined by the equation for that product term. Each output function is provided with output polarity fuses. These fuses permit the designer to configure each output independently to produce either a logic high (by leaving the fuse intact) or a logic low (by programming the fuse) when the equation defining that output is satisfied.

Programming equipment and software make PAL design development quick and easy. Programming is accomplished using TTL voltage levels and is therefore supported by industry standard conventional TTL PLD programmers. After programming and verifying the logic array, an additional security fuse may be programmed to prevent direct copying of proprietary logic designs.

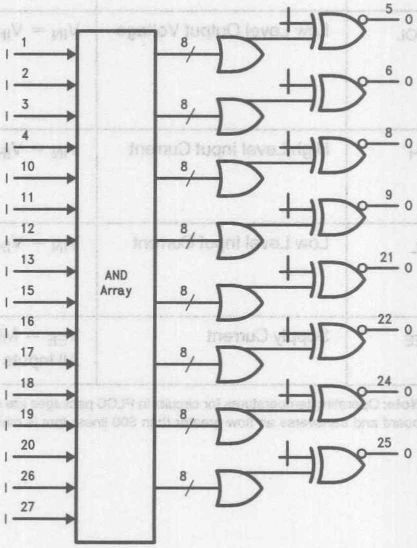
Features

- High speed: t_{PD} = 3 ns max
- Full 28-pin function (all pins used)
- Programmable replacement for ECL logics
- Both 100K and 10 KH I/O compatible versions
- Eight output functions with programmable polarity
- Security fuse to prevent direct copying
- Fully supported by OPAL™ and OPALjr development software
- High density-high performance 28-pin PLCC package

Ordering Information



Block Diagram



TL/L/10712–1

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature under Bias	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
V_{EE} Relative to V_{CC}	−7V to +0.5V
Input Voltage	V_{EE} to +0.5V
Output Current	−50 mA
Lead Temperature (Soldering, 10 Seconds)	1000V
ESD Tolerance	
C_{ZAP} = 100 pF	
R_{ZAP} = 1500Ω	
Test Method: Human Body Model	
Test Specification: NSC SOP-5028	

Recommended Operating Conditions

Symbol	Parameter		Min	Typ	Max	Units
V_{EE}	Supply Voltage	10KH	−5.46	−5.2	−4.94	V
		100K	−4.80	−4.5	−4.20	
T	Operating Temperature (Note)	10KH	0		+75	°C
		100K	0		+85	

Electrical Characteristics Over Recommended Operating Conditions Output Load = 50Ω to −2.0V

Symbol	Parameter	Conditions		T_A	Min	Max	Units
V_{IH}	High Level Input Voltage	Guaranteed Input Voltage High for All Outputs	10KH	0°C +25°C +75°C	−1170 −1130 −1170	−840 −810 −735	mV
			100K	0°C to +85°C	−1165	−880	
V_{IL}	Low Level Input Voltage	Guaranteed Input Voltage Low for All Inputs	10KH	0°C +25°C +75°C	−1950 −1950 −1950	−1480 −1480 −1450	mV
			100K	0°C to +85°C	−1810	−1475	
V_{OH}	High Level Output Voltage	$V_{IN} = V_{IH}$ Max or V_{IL} Min	10KH	0°C +25°C +75°C	−1020 −980 −920	−840 −810 −735	mV
			100K	0°C to +85°C	−1025	−880	
V_{OL}	Low Level Output Voltage	$V_{IN} = V_{IH}$ Max or V_{IL} Min	10KH	0°C +25°C +75°C	−1950 −1950 −1950	−1630 −1630 −1600	mV
			100K	0°C to +85°C	−1810	−1620	
I_{IH}	High Level Input Current	$V_{IN} = V_{IH}$ Max	10KH	0°C +75°C		220	μV
			100K	0°C to +85°C			
I_{IL}	Low Level Input Current	$V_{IN} = V_{IH}$ Min	10KH	0°C +75°C	0.5		μV
			100K	0°C to +85°C			
I_{EE}	Supply Current	V_{EE} = Min All Inputs and Outputs Open	10KH	0°C to +75°C	−230		mA
			100K	0°C to +85°C			

Note: Operating temperatures for circuits in PLCC packages are specified as ambient temperatures (T_A) with circuits in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained.

		(Note 1)		3.0	ns
t_r	Output Rise Time	Measured between 20% and 80% Points	0.25	1.25	ns
t_f	Output Fall Time		0.25	1.25	ns

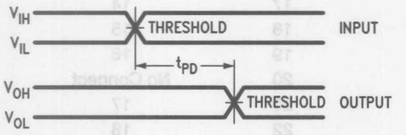
Note 1: All AC Measurements are to be made from Threshold Point.

V_{IH} = Threshold + 400 mV
V_{IL} = Threshold − 400 mV

$$\text{Threshold} = \frac{V_{IH_{Min}} + V_{IL_{Max}}}{2}$$

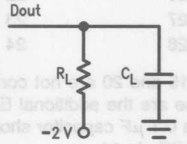
Part	Temp	$V_{IN_{Min}}$	$V_{IL_{Max}}$	Threshold	V_{IH}	V_{IL}
10 kH	0°C	−1170	−1480	−1325	−925	−1725
10 kH	25°C	−1130	−1480	−1300	−900	−1700
10 kH	75°C	−1070	−1450	−1260	−860	−1660
100k	All	−1165	−1475	−1300	−900	−1700

Timing Measurements



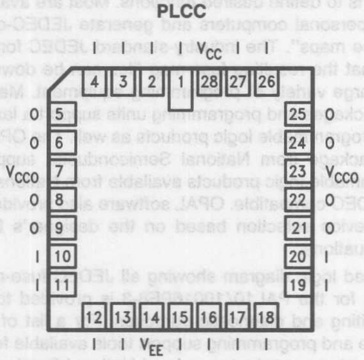
TL/L/10712–2

Test Load



TL/L/10712–3

Connection Diagram



TL/L/10712–4

Top View

Order Number PAL1016PE8-3/PAL10016PE8-3
See NS Package Number V28A

Functional Testing

As with all field-programmable devices, the user of the ECL PAL devices provides the final manufacturing step. While National's PAL devices undergo extensive testing when they are manufactured, their logic function can be fully tested only after they have been programmed to the user's pattern.

To ensure that the programmed PAL devices will operate properly in your system, National Semiconductor (along with most other manufacturers of PAL devices) strongly recommends that devices be functionally tested before being installed in your system. Even though the number of postprogramming functional failures is small, testing the logic function of the PAL devices before they reach system assembly will save board debugging and rework costs. For more information about the functional testing of PAL devices, please refer to National Semiconductor's Application Note #351 and the *Programmable Logic Design Guide.*

Design Development Support

A variety of software tools and programming hardware is available to support the development of designs using PAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate JEDEC-compatible "fuse maps". The industry-standard JEDEC format ensures that the resulting fuse-map files can be downloaded into a large variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

A detailed logic diagram showing all JEDEC fuse-map addresses for the PAL10/10016PE8-3 is provided for direct map editing and diagnostic purposes. For a list of current software and programming support tools available for these devices, please contact your local National Semiconductor sales representative or distributor. If detailed specifications of the ECL PAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support Department.

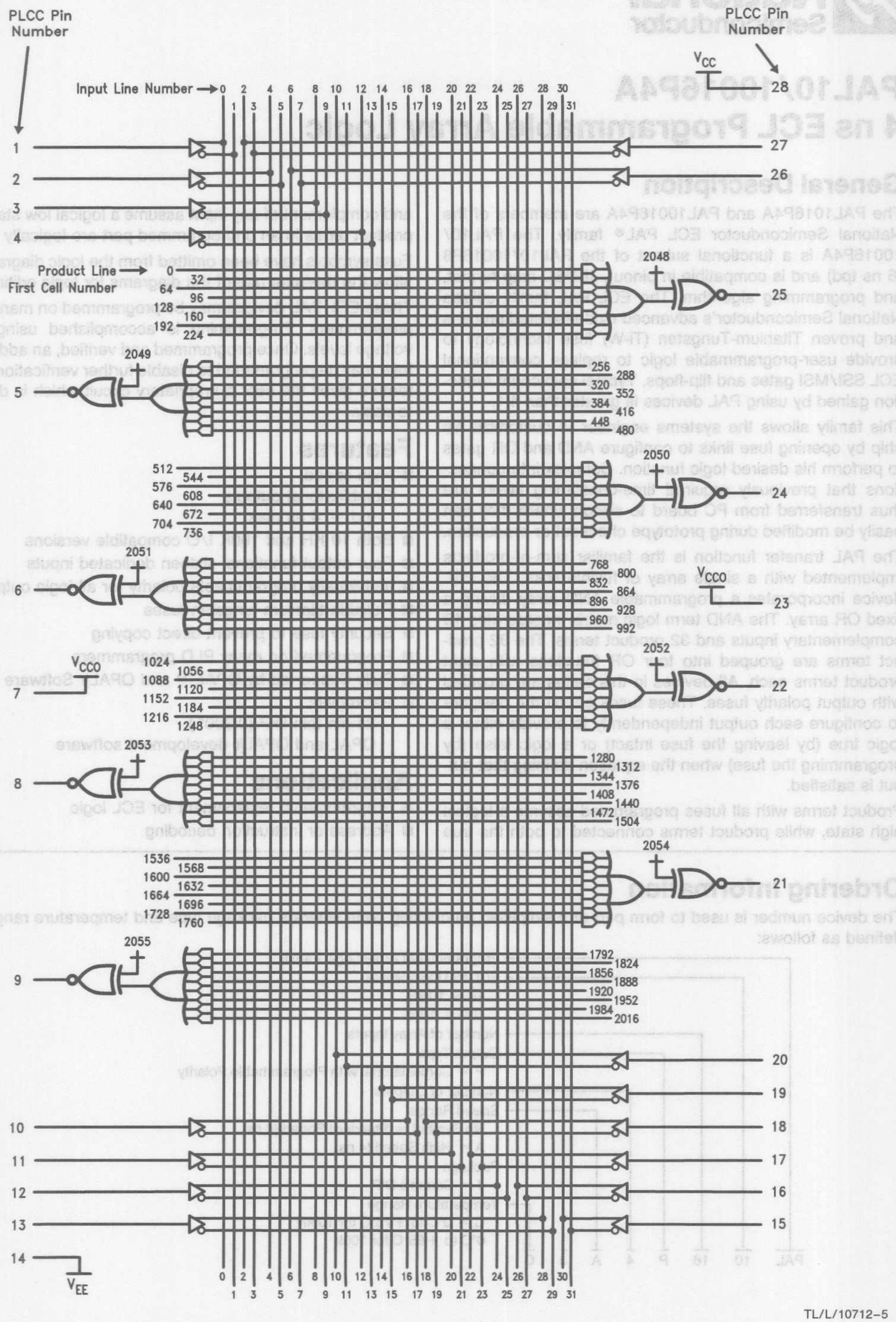
Programming

Most programmers listed below are able to directly program the 28-lead PLCC package. If programming from a DIP socket the following adapter wiring is required:

PLCC Pin	DIP Pin
1	No Connect
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	No Connect
11	9
12	10
13	11
14	12
15	No Connect
16	13
17	14
18	15
19	16
20	No Connect
21	17
22	18
23	19
24	20
25	21
26	22
27	23
28	24

PLCC pins 1, 10, 15 and 20 are not connected to the DIP pins because these are the additional ECL inputs. If using such an adaptor, a 0.1 μF capacitor should be added from PLCC pin 23 to PLCC pin 14.

Logic Diagram—PAL1016PE8-3/PAL10016PE8-3



TL/L/10712-5

JEDEC logic array cell number = product line first cell number + input line number

PAL10/10016P4A
4 ns ECL Programmable Array Logic

General Description

The PAL1016P4A and PAL10016P4A are members of the National Semiconductor ECL PAL® family. The PAL10/10016P4A is a functional subset of the PAL10/10016P8 (6 ns tpd) and is compatible in pinout, JEDEC map format, and programming algorithm. The ECL PAL family utilizes National Semiconductor's advanced oxide-isolated process and proven Titanium-Tungsten (Ti-W) fuse technology to provide user-programmable logic to replace conventional ECL SSI/MSI gates and flip-flops. Typical chip count reduction gained by using PAL devices is greater than 4:1.

This family allows the systems engineer to customize his chip by opening fuse links to configure AND and OR gates to perform his desired logic function. Complex interconnections that previously required time-consuming layout are thus transferred from PC board to silicon where they can easily be modified during prototype checkout or production.

The PAL transfer function is the familiar sum-of-products implemented with a single array of fusible links. The PAL device incorporates a programmable AND array driving a fixed OR array. The AND term logic matrix incorporates 16 complementary inputs and 32 product terms. The 32 product terms are grouped into four OR functions with eight product terms each. All devices in this series are provided with output polarity fuses. These fuses permit the designer to configure each output independently to provide either a logic true (by leaving the fuse intact) or a logic false (by programming the fuse) when the equation defining that output is satisfied.

Product terms with all fuses programmed assume a logical high state, while product terms connected to both the true and complement of any input assume a logical low state. All product terms in an unprogrammed part are logically low.

Fuse symbols have been omitted from the logic diagrams to allow the designer use of the diagrams for logic editing.

These ECL PAL devices may be programmed on many PLD programmers. Programming is accomplished using TTL voltage levels. Once programmed and verified, an additional fuse may be programmed to disable further verification. This feature gives the user a proprietary circuit which is difficult to copy.

Features

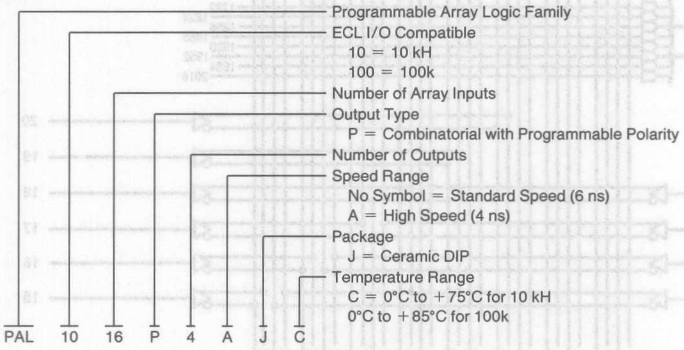
- High speed:
  Combinatorial outputs
  tpd = 4 ns max
- Both 10 KH and 100K I/O compatible versions
- Four output functions; sixteen dedicated inputs
- Individually programmable polarity for all logic outputs
- Reliable titanium-tungsten fuses
- Security fuse to prevent direct copying
- Programmed on many PLD programmers
- Fully Supported by OPAL™ and OPALjr Software
- Packaging:
  24-pin thin DIP (0.300")
  OPAL and OPALjr development software

Applications

- Programmable replacement for ECL logic
- Address or instruction decoding

Ordering Information

The device number is used to form part of a simplified purchasing code where a package type and temperature range are defined as follows:



Office/Distributors for availability and specifications.

Temperature Under Bias	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
V_{EE} Relative to V_{CC}	−7V to +0.5V
Any Input Relative to V_{CC}	V_{EE} to +0.5V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028

Recommended Operating Conditions

Symbol	Parameter		Min	Typ	Max	Units
V_{EE}	Supply Voltage	10 KH	−5.46	−5.2	−4.94	V
		100K	−4.73	−4.5	−4.27	
T	Operating Temperature (Note)	10 KH	0		+75	°C
		100K	0		+85	

DC Electrical Characteristics Over Recommended Operating Conditions

Output Load = 50Ω to −2.0V

Symbol	Parameter	Conditions		T_A	Min	Max	Units
V_{IH}	High Level Input Voltage	Guaranteed Input Voltage High For All Inputs	10 KH	0°C	−1170	−840	mV
				+25°C	−1130	−810	
				+75°C	−1070	−735	
			100K	0°C to +85°C	−1165	−880	
V_{IL}	Low Level Input Voltage	Guaranteed Input Voltage Low For All Inputs	10 KH	0°C	−1950	−1480	mV
				+25°C	−1950	−1480	
				+75°C	−1950	−1450	
			100K	0°C to +85°C	−1810	−1475	
V_{OH}	High Level Output Voltage	V_{IN} = V_{IH} Max. or V_{IL} Min.	10 KH	0°C	−1020	−840	mV
				+25°C	−980	−810	
				+75°C	−920	−735	
			100K	0°C to +85°C	−1025	−880	
V_{OL}	Low Level Output Voltage	V_{IN} = V_{IH} Max. or V_{IL} Min.	10 KH	0°C	−1950	−1630	mV
				+25°C	−1950	−1630	
				+75°C	−1950	−1600	
			100K	0°C to +85°C	−1810	−1620	
I_{IH}	High Level Input Current	V_{IN} = V_{IH} Max.	10 KH	0°C +75°C		220	μA
			100K	0°C to +85°C			
I_{IL}	Low Level Input Current	V_{IN} = V_{IL} Min.	10 KH	0°C +75°C	0.5		μA
			100K	0°C to +85°C			
I_{EE}	Supply Current	V_{EE} = Min. All Inputs and Outputs Open	10 KH	0°C to +75°C	−220		mA
			100K	0°C to +85°C			

Note: Operating temperatures for circuits in Dual-In-Line packages are specified as ambient temperatures (T_A) with circuits in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained. Operating temperatures for circuits packaged in QUAD CERPAK are specified as case temperatures (T_C). All specifications apply after thermal equilibrium has been established.

Switching Characteristics

Over Recommended Operating Conditions, Output load: R_L = 50Ω to −2.0V, C_L = 5 pF to GND

Symbol	Parameter	Measured Test Conditions	Min	Max	Units
t_{PD}	Input to Output	Measured at threshold points (Note 1)		4	ns
t_r	Output Rise Time	Measured between 20% and 80% points	0.5	2.5	ns
t_f	Output Fall Time		0.5	2.5	ns

Note 1: All AC measurements are to be made from threshold point.

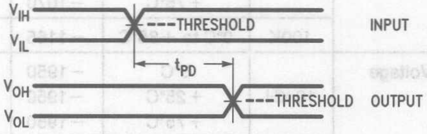
V_{IH} = Threshold + 400 mV

V_{IL} = Threshold − 400 mV

$$\text{Threshold} = \frac{V_{IH_{Min}} + V_{IL_{Max}}}{2}$$

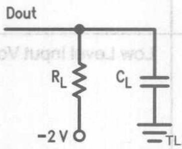
Part	Temp	$V_{IH_{Min}}$	$V_{IL_{Max}}$	Threshold	V_{IH}	V_{IL}
10 kH	−55°C	−1250	−1480	−1365	−965	−1765
10 kH	0°C	−1170	−1480	−1325	−925	−1725
10 kH	25°C	−1130	−1480	−1300	−900	−1700
10 kH	75°C	−1070	−1450	−1260	−860	−1660
10 kH	125°C	−1000	−1420	−1210	−810	−1610
100 k	All	−1165	−1475	−1300	−900	−1700

Timing Measurements



TL/L/9138–7

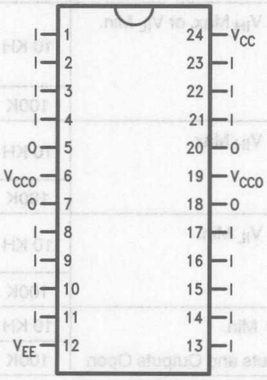
Output Load



TL/L/9138–2

Connection Diagram

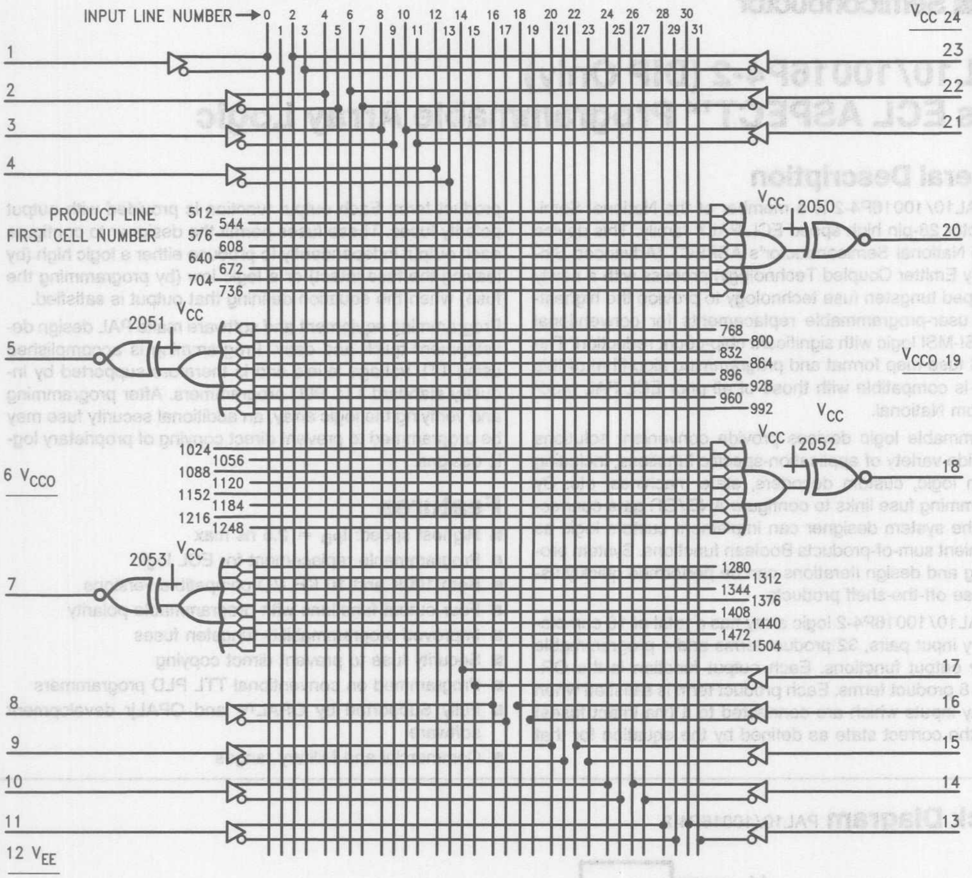
Dual-In-Line Package



TL/L/9138–3

Top View

Logic Diagram PAL1016P4A/PAL10016P4A



TL/L/9138–4

JEDEC logic array cell number = product line first cell number + input line number

Functional Testing

As with all field-programmable devices, the user of the ECL PAL devices provides the final manufacturing step. While National's PAL devices undergo extensive testing when they are manufactured, their logic function can be fully tested only after they have been programmed to the user's pattern.

To ensure that the programmed PAL devices will operate properly in your system, National Semiconductor (along with most other manufacturers of PAL devices) strongly recommends that devices be functionally tested before being installed in your system. Even though the number of post-programming functional failures is small, testing the logic function of the PAL devices before they reach system assembly will save board debugging and rework costs. Refer to National Semiconductor's Application Note #351 and the *Programmable Logic Design Guide* for more information about the functional testing of PAL devices.

Please contact your local sales office for a list of current programming support tools for ECL PAL devices.

National Semiconductor

PAL10/10016P4-2 (DIP Only) 2 ns ECL ASPECT™ Programmable Array Logic

General Description

The PAL10/10016P4-2 is a member of the National Semiconductor 28-pin high speed ECL PAL® family. This device utilizes National Semiconductor's ASPECT (Advanced Single Poly Emitter Coupled Technology) process with a newly developed tungsten fuse technology to provide the highest-speed user-programmable replacements for conventional ECL SSI-MSI logic with significant chip-count reduction. The JEDEC fuse-map format and programming algorithm of this device is compatible with those of all prior ECL PAL products from National.

Programmable logic devices provide convenient solutions for a wide variety of application-specific functions, including random logic, custom decoders, state machines, etc. By programming fuse links to configure AND/OR gate connections, the system designer can implement custom logic as convenient sum-of-products Boolean functions. System prototyping and design iterations can be performed quickly using these off-the-shelf products.

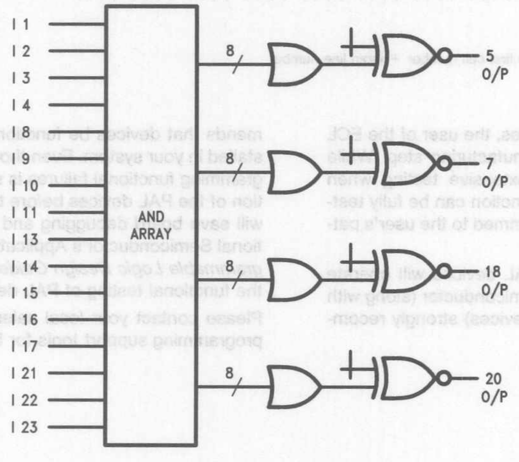
The PAL10/10016P4-2 logic array has a total of 16 complementary input pairs, 32 product terms and 4 programmable polarity output functions. Each output function is the OR-sum of 8 product terms. Each product term is satisfied when all array inputs which are connected to it (via intact fuses) are in the correct state as defined by the equation for that product term. Each output function is provided with output polarity fuses. These fuses permit the designer to configure each output independently to produce either a logic high (by leaving the fuse intact) or a logic low (by programming the fuse) when the equation defining that output is satisfied.

Programming equipment and software make PAL design development quick and easy. Programming is accomplished using TTL voltage levels and is therefore supported by industry standard TTL PLD programmers. After programming and verifying the logic array, an additional security fuse may be programmed to prevent direct copying of proprietary logic designs.

Features

- Highest speed: $t_{PD} = 2.5$ ns max
- Programmable replacement for ECL logic
- Both 100K and 10 KH I/O compatible versions
- Four output functions with programmable polarity
- Improved programmability tungsten fuses
- Security fuse to prevent direct copying
- Programmed on conventional TTL PLD programmers
- Fully Supported by OPAL™ and OPALjr development software
- Commercial and Military ranges

Block Diagram PAL10/10016P4-2



TL/L/10711-1

$V_{EE} = 12$, $V_{CC} = 24$, V_{CCO} (5, 7) = 6
V_{CCO} (18, 20) = 19
Pinout applies to 24-pin DIP

Office/Distributors for availability and specifications.

Temperature Under Bias	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
V_{EE} Relative to V_{CC}	−7V to +0.5V
Input Voltage	V_{EE} to +0.5V

ESD Tolerance	1000V
C_{ZAP} = 100 pF	
R_{ZAP} = 1500Ω	
Test Method: Human Body Model	
Test Specification: NSC SOP-5-028	

Recommended Operating Conditions for Commercial Range

Symbol	Parameter		Min	Typ	Max	Units
V_{EE}	Supply Voltage	10 KH	−5.46	−5.2	−4.94	V
		100K	−4.80	−4.5	−4.20	
T	Operating Temperature (Note)	10 KH	0		+75	°C
		100K	0		+85	

Electrical Characteristics Over Recommended Operating Conditions

Output Load = 50Ω to −2.0V

Symbol	Parameter	Conditions		T_A	Min	Max	Units
V_{IH}	High Level Input Voltage	Guaranteed Input Voltage High For All Inputs	10 KH	0°C +25°C +75°C	−1170 −1130 −1070	−840 −810 −735	mV
			100K	0°C to +85°C	−1165	−880	
V_{IL}	Low Level Input Voltage	Guaranteed Input Voltage Low For All Inputs	10 KH	0°C +25°C +75°C	−1950 −1950 −1950	−1480 −1480 −1450	mV
			100K	0°C to +85°C	−1810	−1475	
V_{OH}	High Level Output Voltage	V_{IN} = V_{IH} Max. or V_{IL} Min.	10 KH	0°C +25°C +75°C	−1020 −980 −920	−840 −810 −735	mV
			100K	0°C to +85°C	−1025	−880	
V_{OL}	Low Level Output Voltage	V_{IN} = V_{IH} Max. or V_{IL} Min.	10 KH	0°C +25°C +75°C	−1950 −1950 −1950	−1630 −1630 −1600	mV
			100K	0°C to +85°C	−1810	−1620	
I_{IH}	High Level Input Current	V_{IN} = V_{IH} Max.	10 KH	0°C to +75°C		220	μA
			100K	0°C to +85°C			
I_{IL}	Low Level Input Current	V_{IN} = V_{IL} Min.	10 KH	0°C to +75°C	0.5		μA
			100K	0°C to +85°C			
I_{EE}	Supply Current	V_{EE} = Min. All Inputs and Outputs Open	10 KH	0°C to +75°C	−220		mA
			100K	0°C to +85°C			

Note: Operating temperatures for circuits in N and J packages are specified as ambient temperatures (T_A) with circuits in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained.

Switching Characteristics

Over Recommended Operating Conditions, Output load: R_L = 50Ω to −2.0V, C_L = 5 pF to GND

Symbol	Parameter	Measured Test Conditions	Commercial		Units
			Min	Max	
t_{PD}	Input to Output	Measured at Threshold Points (Note 1)		2.5	ns
t_r	Output Rise Time	Measured between 20% and 80% points	0.25	1.25	ns
t_f	Output Fall Time		0.25	1.25	ns

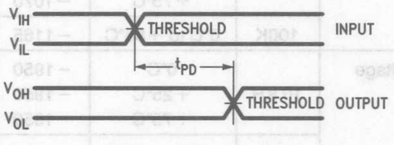
Note 1: All AC Measurements are to be made from Threshold Point.

V_{IH} = Threshold + 400 mV
V_{IL} = Threshold − 400 mV

$$\text{Threshold} = \frac{V_{IH_{Min}} + V_{IL_{Max}}}{2}$$

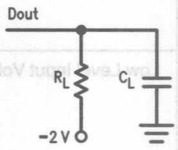
Part	Temp	$V_{IN_{Min}}$	$V_{IL_{Max}}$	Threshold	V_{IH}	V_{IL}
10 kH	−55°C	−1250	−1480	−1365	−965	−1765
10 kH	0°C	−1170	−1480	−1325	−925	−1725
10 kH	25°C	−1130	−1480	−1300	−900	−1700
10 kH	75°C	−1070	−1450	−1260	−860	−1660
10 kH	125°C	−1000	−1420	−1210	−810	−1610
100k	All	−1165	−1475	−1300	−900	−1700

Timing Measurements



TL/L/10711-2

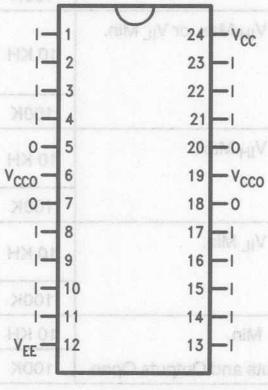
Test Load



TL/L/10711-3

Connection Diagram

Dual-In-Line Package



TL/L/10711-4

Top View

Functional Testing

As with all field-programmable devices, the user of the ECL PAL devices provides the final manufacturing step. While National's PAL devices undergo extensive testing when they are manufactured, their logic function can be fully tested only after they have been programmed to the user's pattern.

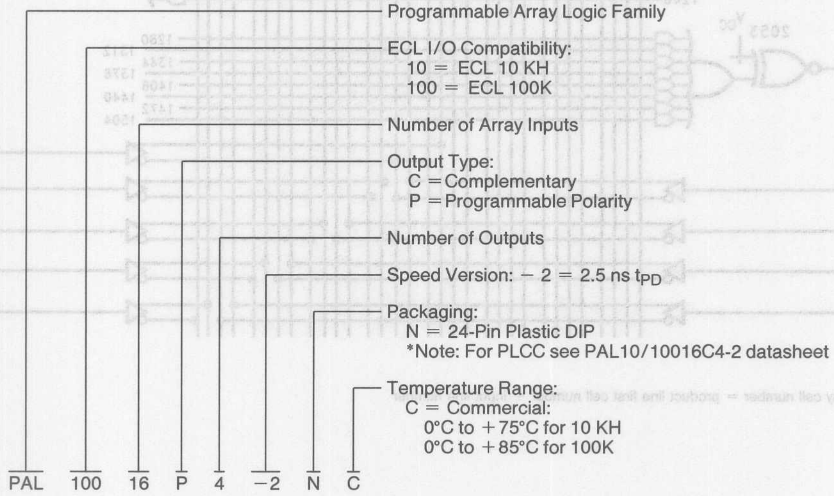
To ensure that the programmed PAL devices will operate properly in your system, National Semiconductor (along with most other manufacturers of PAL devices) strongly recommends that devices be functionally tested before being installed in your system. Even though the number of post-programming functional failures is small, testing the logic function of the PAL devices before they reach system assembly will save board debugging and rework costs. For more information about the functional testing of PAL devices, please refer to National Semiconductor's Application Note #351 and the *Programmable Logic Design Guide.*

Design Development Support

A variety of software tools and programming hardware is available to support the development of designs using PAL products. Typical software packages accept Boolean logic equations to define desired functions. Most are available to run on personal computers and generate JEDEC-compatible "fuse maps". The industry-standard JEDEC format ensures that the resulting fuse-map files can be downloaded into a large variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

A detailed logic diagram showing all JEDEC fuse-map addresses for the PAL10/10016P4-2 is provided for direct map editing and diagnostic purposes. For a list of current software and programming support tools available for these devices, please contact your local National Semiconductor sales representative or distributor. If detailed specifications of the ECL PAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support Department.

Ordering Information



Programmable Array Logic Family

ECL I/O Compatibility:
10 = ECL 10 KH
100 = ECL 100K

Number of Array Inputs

Output Type:
C = Complementary
P = Programmable Polarity

Number of Outputs

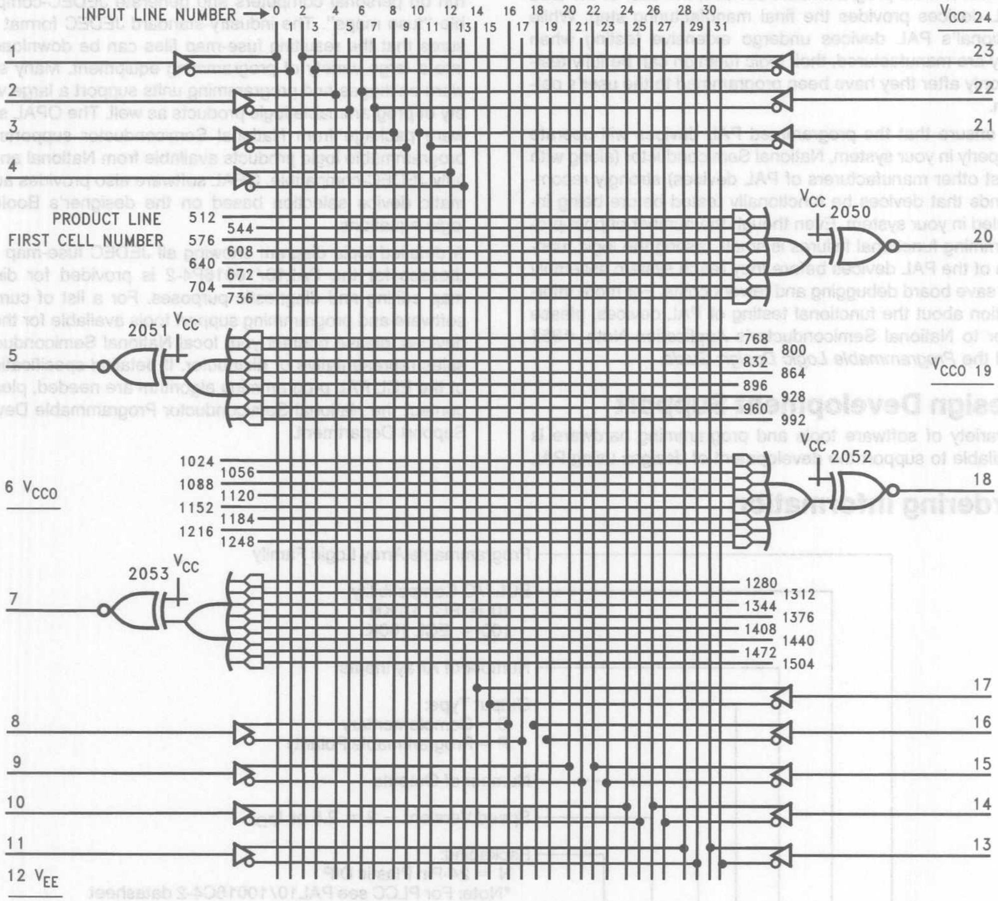
Speed Version: − 2 = 2.5 ns t_{PD}

Packaging:
N = 24-Pin Plastic DIP
*Note: For PLCC see PAL10/10016C4-2 datasheet

Temperature Range:
C = Commercial:
0°C to +75°C for 10 KH
0°C to +85°C for 100K

PAL 100 16 P 4 −2 N C

Logic Diagram—PAL1016P4-2/PAL10016P4-2



TL/L/10711–5

JEDEC logic array cell number = product line first cell number + input line number

National Semiconductor

PAL10/10016C4-2 (PLCC Only) 2 ns ECL ASPECT™ Programmable Array Logic

General Description

The PAL10/10016C4-2 is a member of the National Semiconductor 28-pin high speed ECL PAL® family. This device utilizes National Semiconductor's ASPECT (Advanced Single Poly ECL Technology) Process with a newly developed tungsten fuse technology to provide the highest-speed user-programmable replacements for conventional ECL SSI-MSI logic with significant chip-count reduction. The JEDEC fuse-map format and programming algorithm of this device is compatible with those of all prior ECL PAL products from National.

Programmable logic devices provide convenient solutions for a wide variety of application-specific functions, including random logic, custom decoders, state machines, etc. By programming fuse links to configure AND/OR gate connections, the system designer can implement custom logic as convenient sum-of-products Boolean functions. System prototyping and design iterations can be performed quickly using these off-the-shelf products.

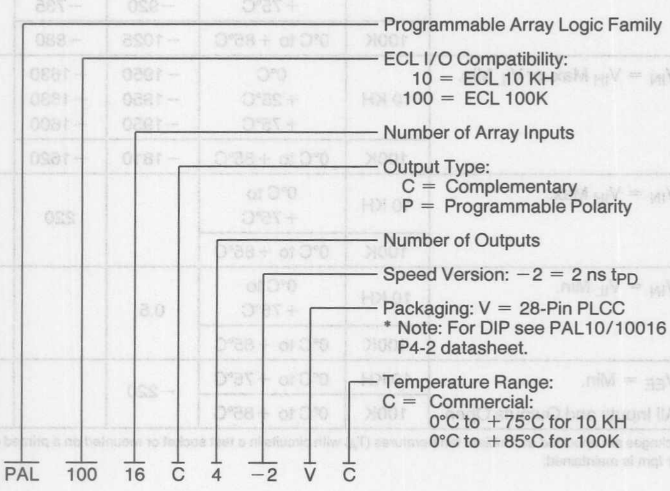
The PAL10/10016C4-2 logic array has a total of 16 complementary input pairs, 32 product terms and 4 complementary output functions. Each output function is the OR-sum of 8 product terms. Each product term is satisfied when all array inputs which are connected to it (via intact fuses) are in the correct state. Complementary outputs eliminate the need for external inverters and allow for more convenient output OR-tying. They are also suitable for differential sensing for increased noise immunity. All input pins have on-chip 50 kΩ pull-down resistors.

Programming equipment and software make PAL design development quick and easy. Programming is accomplished using TTL voltage levels and is therefore supported by several conventional TTL PLD programming units. After programming and verifying the logic array, an additional security fuse may be programmed to prevent direct copying of proprietary logic designs.

Features

- Highest speed: t_{PD} = 2 ns max
- Full 28-pin function
- Programmable replacement for ECL logic
- Both 100K and 10 KH I/O compatible versions
- Four output functions with complementary outputs
- Improved programmability tungsten fuses
- Security fuse to prevent direct copying
- Programmed on conventional TTL PLD programmers
- Fully Supported by OPAL™ and OPALjr development software
- High density-High performance 28-pin PLCC package

Ordering Information



Programmable Array Logic Family

ECL I/O Compatibility:
10 = ECL 10 KH
100 = ECL 100K

Number of Array Inputs

Output Type:
C = Complementary
P = Programmable Polarity

Number of Outputs

Speed Version: −2 = 2 ns t_{PD}

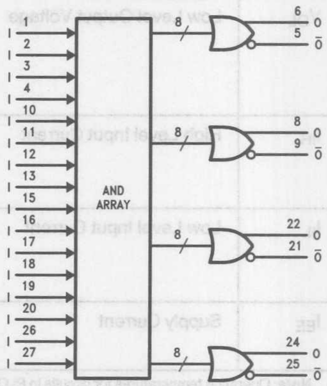
Packaging: V = 28-Pin PLCC
* Note: For DIP see PAL10/10016 P4-2 datasheet.

Temperature Range:
C = Commercial:
0°C to +75°C for 10 KH
0°C to +85°C for 100K

PAL 100 16 C 4 −2 V C

Block Diagram

PAL10/10016C4-2



TL/L/10454-2

V_{EE} = 14, V_{CC} = 28, V_{CCO} (5, 6, 8, 9) = 7
V_{CCO} (21, 22, 24, 25) = 23
Pinout applies to 28-pin PLCC

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Temperature Under Bias	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
V_{EE} Relative to V_{CC}	−7V to +0.5V
Input Voltage	V_{EE} to +0.5V
Output Current	−50 mA
Lead Temperature (Soldering, 10 seconds)	300°C
ESD Tolerance	1000V

C_{ZAP} = 100 pF
R_{ZAP} = 1500Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028

Recommended Operating Conditions

Symbol	Parameter		Min	Typ	Max	Units
V_{EE}	Supply Voltage	10 KH	−5.46	−5.2	−4.94	V
		100K	−4.80	−4.5	−4.20	
T	Operating Temperature (Note)	10 KH	0		+75	°C
		100K	0		+85	

Electrical Characteristics Over Recommended Operating Conditions

Output Load = 50Ω to −2.0V

Symbol	Parameter	Conditions		T_A	Min	Max	Units
V_{IH}	High Level Input Voltage	Guaranteed Input Voltage High For All Inputs	10 KH	0°C	−1170	−840	mV
				+25°C	−1130	−810	
				+75°C	−1070	−735	
			100K	0°C to +85°C	−1165	−880	
V_{IL}	Low Level Input Voltage	Guaranteed Input Voltage Low For All Inputs	10 KH	0°C	−1950	−1480	mV
				+25°C	−1950	−1480	
				+75°C	−1950	−1450	
			100K	0°C to +85°C	−1810	−1475	
V_{OH}	High Level Output Voltage	$V_{IN} = V_{IH}$ Max. or V_{IL} Min.	10 KH	0°C	−1020	−840	mV
				+25°C	−980	−810	
				+75°C	−920	−735	
			100K	0°C to +85°C	−1025	−880	
V_{OL}	Low Level Output Voltage	$V_{IN} = V_{IH}$ Max. or V_{IL} Min.	10 KH	0°C	−1950	−1630	mV
				+25°C	−1950	−1630	
				+75°C	−1950	−1600	
			100K	0°C to +85°C	−1810	−1620	
I_{IH}	High Level Input Current	$V_{IN} = V_{IH}$ Max.	10 KH	0°C to +75°C		220	μA
			100K	0°C to +85°C			
I_{IL}	Low Level Input Current	$V_{IN} = V_{IL}$ Min.	10 KH	0°C to +75°C	0.5		μA
			100K	0°C to +85°C			
I_{EE}	Supply Current	V_{EE} = Min. All Inputs and Outputs Open	10 KH	0°C to +75°C	−220		mA
			100K	0°C to +85°C			

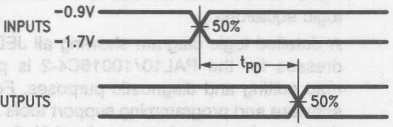
Note: Operating temperatures for circuits in PLCC packages are specified as ambient temperatures (T_A) with circuits in a test socket or mounted on a printed circuit board and transverse air flow greater than 500 linear fpm is maintained.

Switching Characteristics

Over Recommended Operating Conditions, Output load: $R_L = 50\Omega$ to $-2.0V$, $C_L = 5$ pF to GND

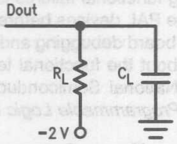
Symbol	Parameter	Measured Test Conditions	Min	Max	Units
t_{PD}	Input to Output	Measured at 50% points		2.0	ns
t_r	Output Rise Time	Measured between 20% and 80% points	0.25	1.25	ns
t_f	Output Fall Time		0.25	1.25	ns

Timing Measurements



TL/L/10454-3

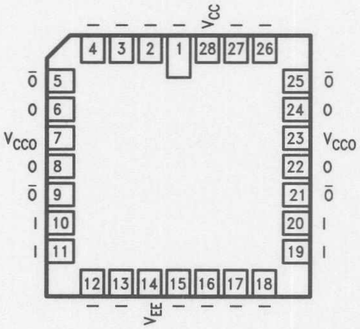
Test Load



TL/L/10454-4

Connection Diagram

PLCC



TL/L/10454-5

Top View

Order Number PAL1016C4-2/PAL10016C4-2
See NS Package Number V28A

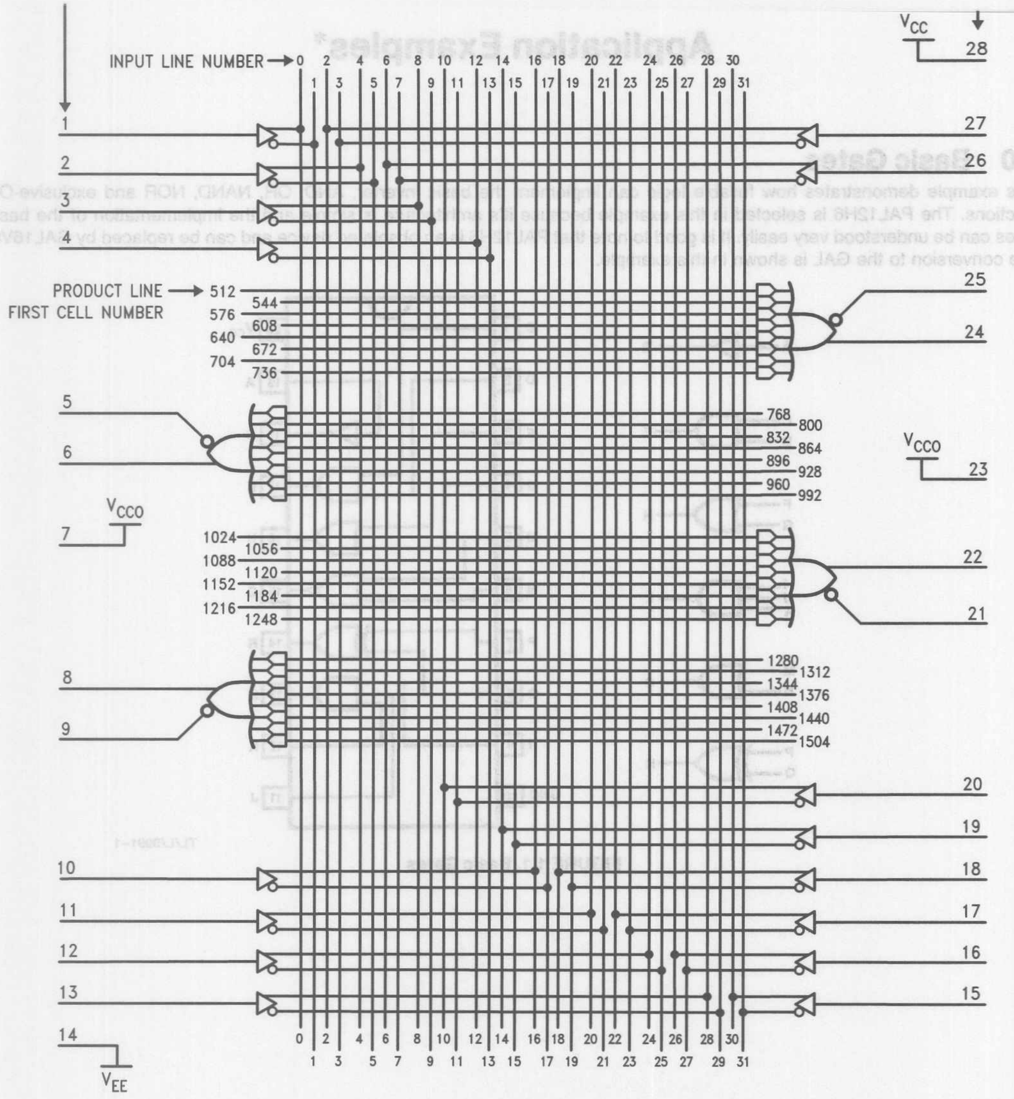
National's PAL devices undergo extensive testing when they are manufactured, their logic function can be fully tested only after they have been programmed to the user's pattern.

To ensure that the programmed PAL devices will operate properly in your system, National Semiconductor (along with most other manufacturers of PAL devices) strongly recommends that devices be functionally tested before being installed in your system. Even though the number of post-programming functional failures is small, testing the logic function of the PAL devices before they reach system assembly will save board debugging and rework costs. For more information about the functional testing of PAL devices, please refer to National Semiconductor's Application Note #351 and the *Programmable Logic Design Guide.*

Design Development Support

A variety of software tools and programming hardware is available to support the development of designs using PAL run on personal computers and generate JEDEC-compatible "fuse maps". The industry-standard JEDEC format ensures that the resulting fuse-map files can be downloaded into a large variety of programming equipment. Many software packages and programming units support a large variety of programmable logic products as well. The OPAL software package from National Semiconductor supports all programmable logic products available from National and is fully JEDEC-compatible. OPAL software also provides automatic device selection based on the designer's Boolean logic equations.

A detailed logic diagram showing all JEDEC fuse-map addresses for the PAL10/10016C4-2 is provided for direct map editing and diagnostic purposes. For a list of current software and programming support tools available for these devices, please contact your local National Semiconductor sales representative or distributor. If detailed specifications of the ECL PAL programming algorithm are needed, please contact the National Semiconductor Programmable Device Support Department.



TL/L/10454-6

JEDEC logic array cell number = product line first cell number + input line number

10016C4-2

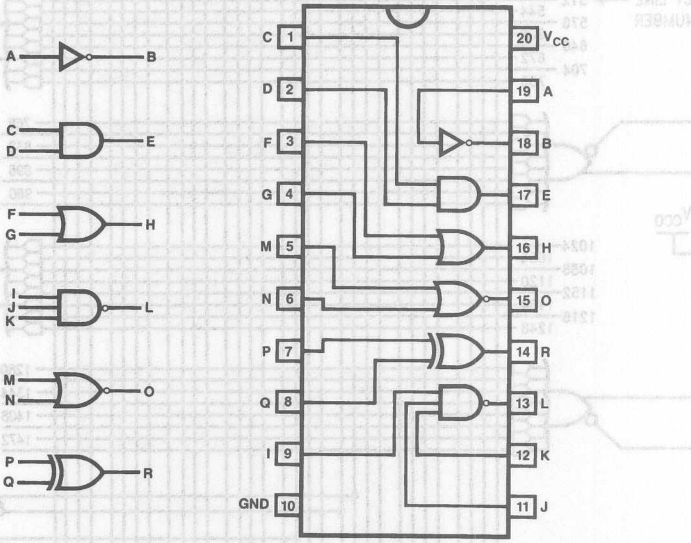
2

National Semiconductor

Application Examples*

1.0 Basic Gates

This example demonstrates how fusable logic can implement the basic inverter, AND, OR, NAND, NOR and exclusive-OR functions. The PAL12H6 is selected in this example because it's architecture is simple and the implementation of the basic gates can be understood very easily. It is good to note that PAL12H6 is an obsoleted device and can be replaced by GAL16V8. The conversion to the GAL is shown in this example.



TL/L/9991-1

FIGURE 1.1. Basic Gates

*Applications contained in this section are for illustration purposes only and National makes no representation or warranty that such applications will be suitable for the use specified without further testing or modification.

1.0 Basic Gates (Continued)

OPAL™ INPUT FILE

```
title    Basic gate
pattern  GATES
revision A
author   Tarif Engineer
company  National Semiconductor Corporation
Date     11/28/1989

chip GATES PAL12H6

; pin 1    2    3    4    5    6    7    8    9    10
      C    D    F    G    M    N    P    Q    I    GND
; pin 11   12   13   14   15   16   17   18   19   20
      J    K    L    R    O    H    E    B    A    VCC
equations

     B = /A
     E = C * D
     H = F + G
     L = /I + /J + /K
     O = /M * /N
     R = P * /Q
        + /P * Q


; end of GATES
```

TL/L/9991-2

OPAL™ JEDEC FILE

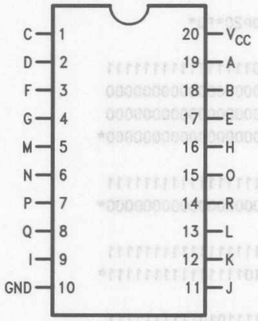
```
PAL12H6
title    Basic gate
pattern  GATES
revision A
author   Tarif Engineer
company  National Semiconductor Corporation
Date     11/28/1989
*
QF0384*QP20*F0*
L0000
111111101111111111111111
000000000000000000000000
000000000000000000000000
000000000000000000000000*
L0096
010111111111111111111111
000000000000000000000000*
L0144
111101111111111111111111
111111101111111111111111*
L0192
111111111110101111111111
000000000000000000000000*
L0240
111111111111111011011111
111111111111111100111111*
L0288
111111111111111111111011
111111111111111111111110
111111111111111111101111
000000000000000000000000*
C1BB9*
0000
```

TL/L/9991-4

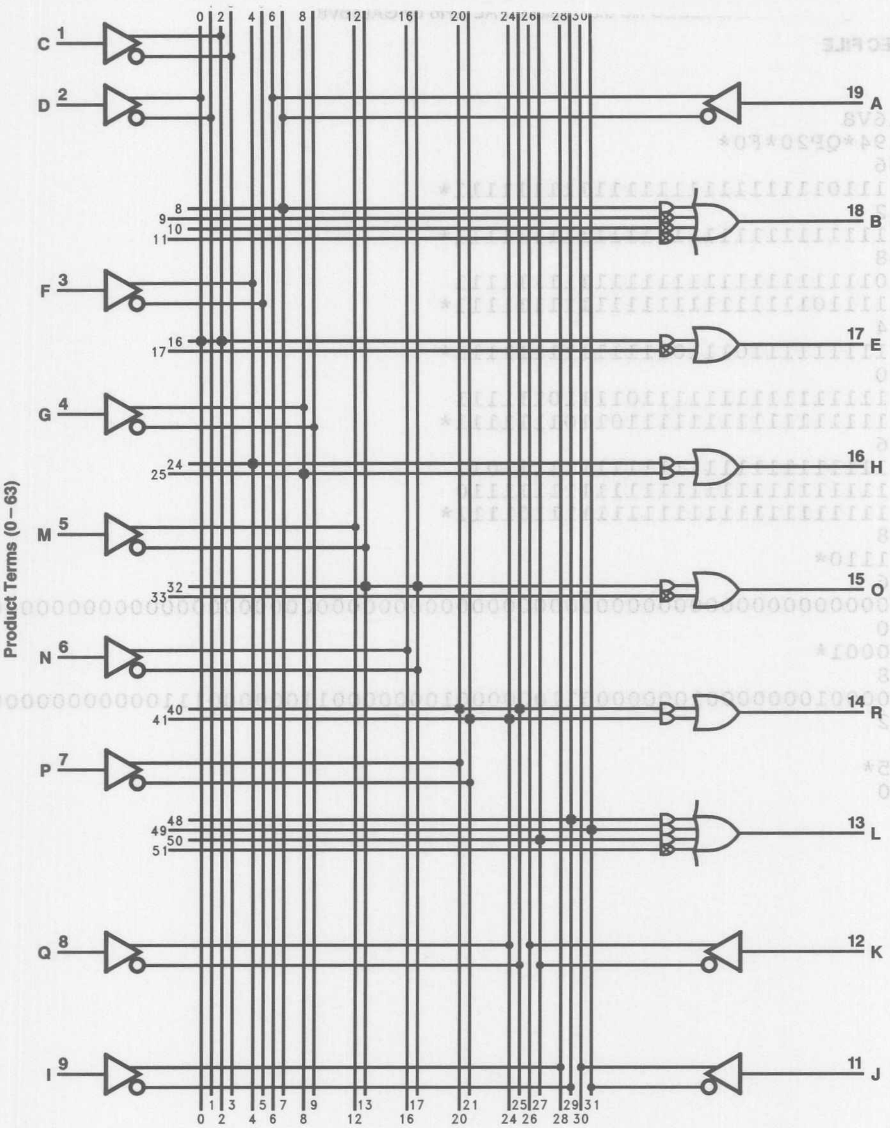
```
Pin   Label                             Type
---   -----                             ----
1     C                                 com input
2     D                                 com input
3     F                                 com input
4     G                                 com input
5     M                                 com input
6     N                                 com input
7     P                                 com input
8     Q                                 com input
9     I                                 com input
10    GND                               ground pin
11    J                                 com input
12    K                                 com input
13    L                                 pos,com output
14    R                                 pos,com output
15    O                                 pos,com output
16    H                                 pos,com output
17    E                                 pos,com output
18    B                                 pos,com output
19    A                                 com input
20    VCC                               power pin
```

TL/L/9991-6

Chip Diagram (DIP)



TL/L/9991-35



TL/L/9991–30

FIGURE 1.2. PAL12H6 Logic Diagram Showing Fuse Pattern of Basic Gates Example

1.0 Basic Gates (Continued)

The previous basic gates example can be implemented using GAL16V8 as the target device and the conversion can be done by using PAL2GAL utility offered by OPAL or OPAL$_{jr}$ software.

PAL2GAL will generate a GAL JEDEC file that replace PAL12H6 by GAL16V8.

GAL JEDEC FILE

```
☻
GAL16V8
QF2194*QP20*F0*
L0256
11111101111111111111111111111111*
L0512
01011111111111111111111111111111*
L0768
11110111111111111111111111111111
11111110111111111111111111111111*
L1024
11111111111110111011111111111111*
L1280
11111111111111111111011110111111
11111111111111111111101101111111*
L1536
11111111111111111111111111111011
11111111111111111111111111111110
11111111111111111111111111101111*
L2048
01111110*
L2056
0000000000000000000000000000000000000000000000000000000000000000*
L2120
10000001*
L2128
0000000010000000100000001100000010000000110000001110000000000000*
L2192
10*
C2755*
♥0000
```

TL/L/9991–7

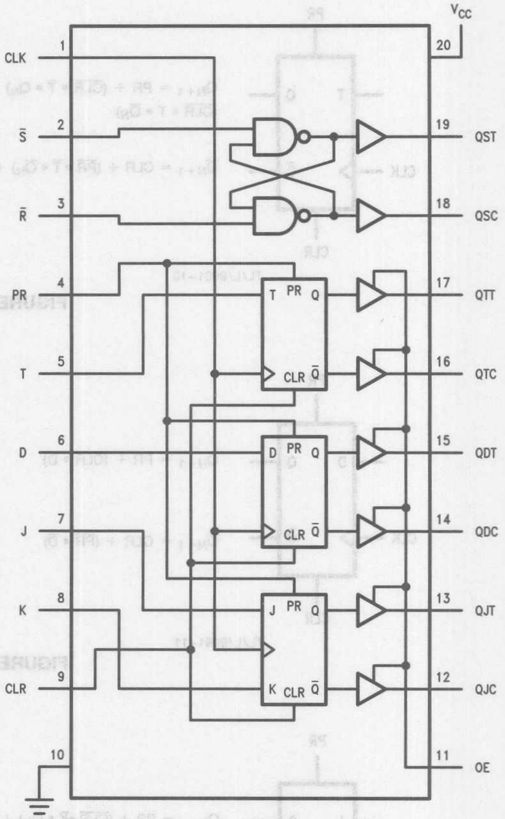
2.0 Basic Flip-Flops

DESCRIPTION

In the Basic Gates application on the preceding pages, each 'gate' was directly connected to an output pin. Here, the output registers of the GAL16V8 are used. A simple RS latch, a T (Toggle) flip-flop, a D flip-flop, and a JK flip-flop are incorporated into a GAL16V8 *(Figure 2.1)*. Each is shown with its truth table and defining equations in *Figures 2.2–2.5*. Note that all 3 flip-flops have synchronous preset (PR) and clear (CLR) inputs, while the RS latch does not. Also, the RS latch is not connected to the clock input; this was done to show the versatility of the GAL16V8—with a GAL device, the user is not locked in to a specific architecture.

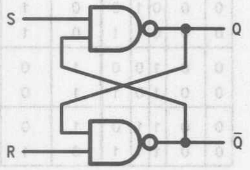
The CUPL design input file *(Figure 2.6)* and simulation file *(Figure 2.7)* are constructed by the designer. Each output must be given a distinct name, and any clocked circuit must be denoted with an appropriate extension (.D) in the logic equations. The simulation file is again provided for design verification.

This example has some subtle requirements that may not be apparent to the first-time user. When the RS latch is not being tested, it must remain in its latched state with the output levels specified, or with the variable N (not tested) specified instead. Also, when executing a preset or clear, remember that it will affect all flip-flops; even those not being tested will still respond. Finally, all output levels should be specified or marked with the variable N; the variable X, which indicates a 'don't care' condition, will not suffice.



TL/L/9991-8

FIGURE 2.1. Basic Flip-Flops Pinout



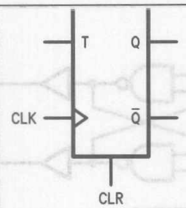
$Q_{N+1} = \overline{S} + (R \bullet Q_N)$

$\overline{Q}_{N+1} = \overline{R} + (S \bullet \overline{Q}_N)$

TL/L/9991-9

S	R	Q_N	Q_{N+1}	$\overline{Q}_{N+1}$	Comments
0	0	0	1	1	Invalid
0	0	1	1	1	
0	1	0	1	0	Set
0	1	1	1	0	
1	0	0	0	1	Reset
1	0	1	0	1	
1	1	0	0	1	Latch
1	1	1	1	0	

FIGURE 2.2. RS Latch



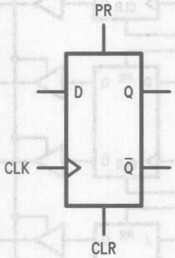
$Q_{N+1} = PR + (\overline{CLR} \bullet \overline{T} \bullet Q_N) + (\overline{CLR} \bullet T \bullet \overline{Q}_N)$

$\overline{Q}_{N+1} = CLR + (\overline{PR} \bullet \overline{T} \bullet \overline{Q}_N) + (\overline{PR} \bullet T \bullet Q_N)$

TL/L/9991-10

1	0	X	X	1	0	Preset
0	1	X	X	0	1	Clear
0	0	0	0	0	1	Hold
0	0	0	1	1	0	
0	0	1	0	1	0	Toggle
0	0	1	1	0	1	

FIGURE 2.3. T Flip-Flop



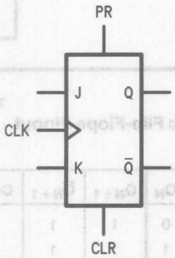
$Q_{N+1} = PR + (\overline{CLR} \bullet D)$

$\overline{Q}_{N+1} = CLR + (\overline{PR} \bullet \overline{D})$

TL/L/9991-11

PR	CLR	D	Q_N	Q_{N+1}	$\overline{Q}_{N+1}$	Comments
1	1	X	X	1	1	Invalid
1	0	X	X	1	0	Preset
0	1	X	X	0	1	Clear
0	0	0	0	0	1	Reset
0	0	0	1	0	1	
0	0	1	0	1	0	Set
0	0	1	1	1	0	

FIGURE 2.4. D Flip-Flop



$Q_{N+1} = PR + (\overline{CLR} \bullet \overline{K} \bullet Q_N) + (\overline{CLR} \bullet J \bullet \overline{Q}_N)$

$\overline{Q}_{N+1} = CLR + (\overline{PR} \bullet \overline{J} \bullet \overline{Q}_N) + (\overline{PR} \bullet K \bullet Q_N)$

TL/L/9991-12

PR	CLR	J	K	Q_N	Q_{N+1}	$\overline{Q}_{N+1}$	Comments
1	1	X	X	X	1	1	Invalid
1	0	X	X	X	1	0	Preset
0	1	X	X	X	0	1	Clear
0	0	0	0	0	0	1	Hold
0	0	0	0	1	1	0	
0	0	0	1	0	0	1	Reset
0	0	0	1	1	0	1	
0	0	1	0	0	1	0	Set
0	0	1	0	1	1	0	
0	0	1	1	0	1	0	Toggle
0	0	1	1	1	0	1	

FIGURE 2.5. JK Flip-Flop

```
/*                                                            */
/*                      CUPL INPUT FILE                       */
/*      Flip-flops and latches implemented in a GAL16V8       */
/*                                                            */
/**************************************************************/
          PARTNO          456STX;
          NAME            FLIPFLOP;
          REV             1;
          DATE            4/11/86;
          DESIGNER        Joe Engineer;
          COMPANY         National Semiconductor;
          ASSEMBLY        Clock Board;
          LOCATION        U238;

/* RS latch */

          pin [2,3,19,18] = [S,R,QST,QSC];

/* T flip-flop */

          pin [5,17,16] = [T,QTT,QTC];

/* D flip-flop */

          pin [6,15,14] = [D,QDT,QDC];

/* JK flip-flop */

          pin [7,8,13,12] = [J,K,QJT,QJC];

/* control */

          pin [1,4,9,11] = [CLK,PR,CLR,OE];

/* logic equations */

          /* RS latch */

                    QST = !S # (R & QST);
                    QSC = !R # (S & QSC);

          /* T flip-flop */

                    QTT.D = PR # (!CLR & !T & QTT) # (!CLR & T & QTC);
                    QTC.D = CLR # (!PR & !T & QTC) # (!PR & T & QTT);

          /* D flip-flop */

                    QDT.D = PR # (D & !CLR);
                    QDC.D = CLR # (!D & !PR);

          /* JK flip-flop */

                    QJT.D = PR # (J & QJC & !CLR) # (!K & QJT & !CLR);
                    QJC.D = CLR # (!J & QJC & !PR) # (K & QJT & !PR);
```

TL/L/9991-15

FIGURE 2.6. CUPL Input File

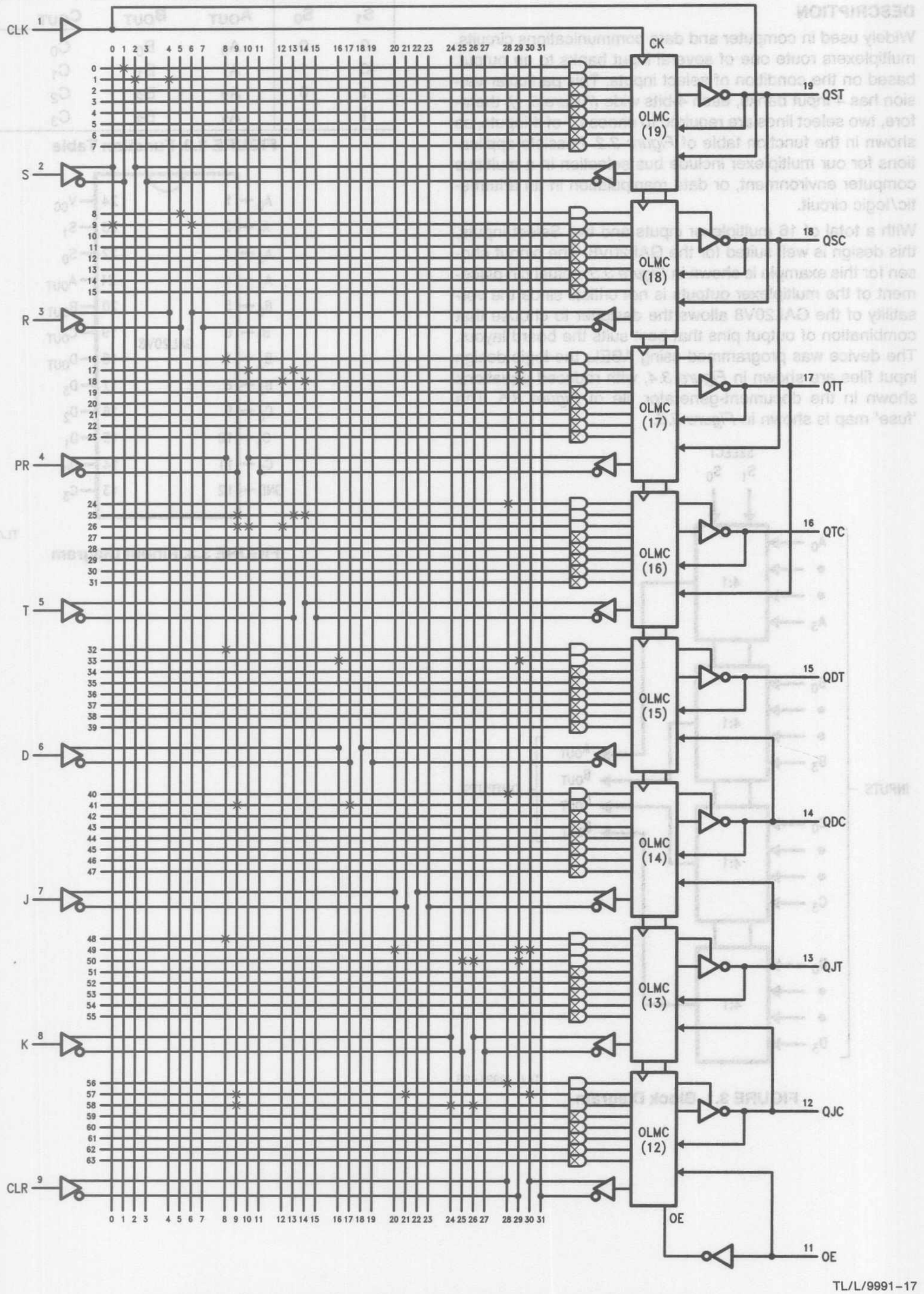
2.0 Basic Flip-Flops (Continued)

```
/******************************************************************/
/*                                                                */
/*                    CUPL SIMULATION FILE                        */
/*       Flip-flops and latches implemented in a GAL16V8          */
/*                                                                */
/******************************************************************/
          PARTNO          456STX;
          NAME            FLIPFLOP;
          REV             1;
          DATE            4/11/86;
          DESIGNER        Joe Engineer;
          COMPANY         National Semiconductor;
          ASSEMBLY        Clock Board;
          LOCATION        U238;

/* The Order statement specifies the layout of the vector table.
          %n = n spaces inserted between variables.                  */

order:    OE,%1,CLK,%2,S,R,%1,QST,QSC,%2,PR,%1,CLR,%2,
          T,%1,QTT,QTC,%2,D,%1,QDT,QDC,%2,J,K,%1,QJT,QJC;

vectors:
/*        RS-latch                T-FF        D-FF        JK-FF    */
/*OE CLK  SR QSTQSC    PR CLR   T QTTQTC   D QDTQDC   JK QJTQJC */
   0  X   01  H  L      X  X    X  X  X    X  X  X    XX  X  X  /* set */
   0  X   10  L  H      X  X    X  X  X    X  X  X    XX  X  X  /* reset */
   0  X   11  L  H      X  X    X  X  X    X  X  X    XX  X  X  /* latch */
   0  X   10  L  H      X  X    X  X  X    X  X  X    XX  X  X  /* reset */
   0  X   01  H  L      X  X    X  X  X    X  X  X    XX  X  X  /* set */
   0  X   11  H  L      X  X    X  X  X    X  X  X    XX  X  X  /* latch */

   0  C   11  N  N      1  0    X  H  L    X  N  N    XX  N  N  /* preset */
   0  C   11  N  N      0  1    X  L  H    X  N  N    XX  N  N  /* clear */
   0  C   11  N  N      0  0    0  L  H    X  X  X    XX  X  X  /* hold  */
   0  C   11  N  N      0  0    1  H  L    X  X  X    XX  X  X  /* toggle */
   0  C   11  N  N      0  0    0  H  L    X  X  X    XX  X  X  /* hold  */
   0  C   11  N  N      0  0    1  L  H    X  X  X    XX  X  X  /* toggle */
   0  C   11  N  N      0  0    1  H  L    X  X  X    XX  X  X  /* toggle */
   0  C   11  N  N      1  0    X  N  N    X  H  L    XX  N  N  /* preset */
   0  C   11  N  N      0  1    X  N  N    X  L  H    XX  N  N  /* clear */
   0  C   11  N  N      0  0    X  X  X    0  L  H    XX  X  X
   0  C   11  N  N      0  0    X  X  X    1  H  L    XX  X  X  /* test */
   0  C   11  N  N      0  0    X  X  X    1  H  L    XX  X  X
   0  C   11  N  N      0  0    X  X  X    0  L  H    XX  X  X

   0  C   11  N  N      1  0    X  N  N    X  N  N    XX  H  L  /* preset */
   0  C   11  N  N      0  1    X  N  N    X  N  N    XX  L  H  /* clear */
   0  C   11  N  N      0  0    X  X  X    X  X  X    01  L  H
   0  C   11  N  N      0  0    X  X  X    X  X  X    00  L  H  /* hold */
   0  C   11  N  N      0  0    X  X  X    X  X  X    11  H  L  /* toggle */
   0  C   11  N  N      0  0    X  X  X    X  X  X    10  H  L
   0  C   11  N  N      0  0    X  X  X    X  X  X    00  H  L  /* hold */
   0  C   11  N  N      0  0    X  X  X    X  X  X    11  L  H  /* toggle */
   0  C   11  N  N      0  0    X  X  X    X  X  X    10  H  L
   0  C   11  N  N      0  0    X  X  X    X  X  X    01  L  H
```

TL/L/9991-16

FIGURE 2.7. CUPL Simulation File

2.0 Basic Flip-Flops (Continued)

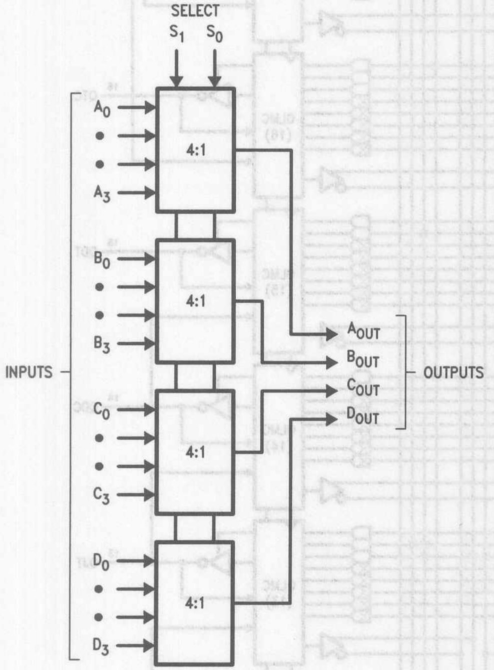


TL/L/9991-17

FIGURE 2.8. GAL16V8 Logic Diagram Showing Basic Flip-Flops Pattern

widely used in computer and data communications circuits, multiplexers route one of several input banks to an output, based on the condition of select inputs. This particular version has 4 input banks, each 4-bits wide *(Figure 3.1)*; therefore, two select lines are required to choose 1 of 4 inputs, as shown in the function table of *Figure 3.2*. Possible applications for our multiplexer include bus selection in a multibus computer environment, or data manipulation in an arithmetic/logic circuit.

With a total of 16 multiplexer inputs and two Select inputs, this design is well suited for the GAL20V8. The pinout chosen for this example is shown in *Figure 3.3*; actual pin placement of the multiplexer outputs is not critical since the versatility of the GAL20V8 allows the designer to choose that combination of output pins that best suits the board layout. The device was programmed using ABEL; the logic design input files are shown in *Figure 3.4*, with reduced equations shown in the document-generator file of *Figure 3.5*. The 'fuse' map is shown in *Figure 3.6*.

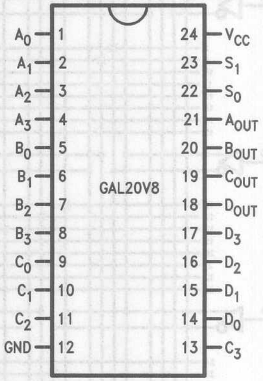


TL/L/9991-23

FIGURE 3.1. Block Diagram

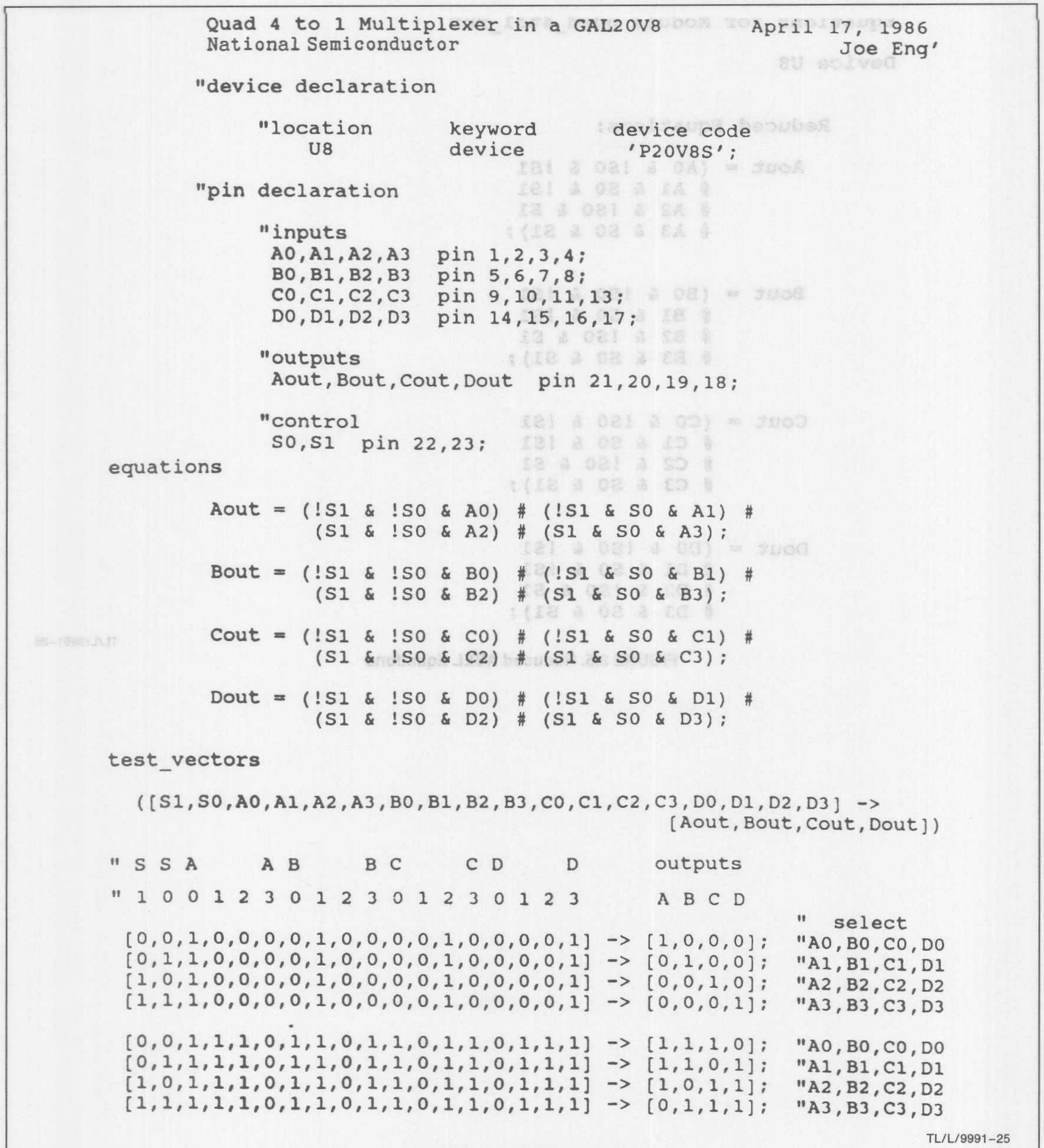
0	0	A_0	B_0	C_0	D_0
0	1	A_1	B_1	C_1	D_1
1	0	A_2	B_2	C_2	D_2
1	1	A_3	B_3	C_3	D_3

FIGURE 3.2. Function Table



TL/L/9991-24

FIGURE 3.3. Pinout Diagram



```
            Quad 4 to 1 Multiplexer in a GAL20V8          April 17, 1986
            National Semiconductor                              Joe Eng'
           "device declaration

                "location        keyword       device code
                    U8           device          'P20V8S';

           "pin declaration

                "inputs
                 A0,A1,A2,A3   pin 1,2,3,4;
                 B0,B1,B2,B3   pin 5,6,7,8;
                 C0,C1,C2,C3   pin 9,10,11,13;
                 D0,D1,D2,D3   pin 14,15,16,17;

                "outputs
                 Aout,Bout,Cout,Dout  pin 21,20,19,18;

                "control
                 S0,S1  pin 22,23;
   equations

            Aout = (!S1 & !S0 & A0) # (!S1 & S0 & A1) #
                   (S1 & !S0 & A2) # (S1 & S0 & A3);

            Bout = (!S1 & !S0 & B0) # (!S1 & S0 & B1) #
                   (S1 & !S0 & B2) # (S1 & S0 & B3);

            Cout = (!S1 & !S0 & C0) # (!S1 & S0 & C1) #
                   (S1 & !S0 & C2) # (S1 & S0 & C3);

            Dout = (!S1 & !S0 & D0) # (!S1 & S0 & D1) #
                   (S1 & !S0 & D2) # (S1 & S0 & D3);

   test_vectors

     ([S1,S0,A0,A1,A2,A3,B0,B1,B2,B3,C0,C1,C2,C3,D0,D1,D2,D3] ->
                                                  [Aout,Bout,Cout,Dout])

   " S S A     A B     B C     C D     D       outputs

   " 1 0 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3       A B C D
                                                                "  select
    [0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1] -> [1,0,0,0];    "A0,B0,C0,D0
    [0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1] -> [0,1,0,0];    "A1,B1,C1,D1
    [1,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1] -> [0,0,1,0];    "A2,B2,C2,D2
    [1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1] -> [0,0,0,1];    "A3,B3,C3,D3

    [0,0,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1] -> [1,1,1,0];    "A0,B0,C0,D0
    [0,1,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1] -> [1,1,0,1];    "A1,B1,C1,D1
    [1,0,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1] -> [1,0,1,1];    "A2,B2,C2,D2
    [1,1,1,1,1,0,1,1,0,1,1,0,1,1,0,1,1,1] -> [0,1,1,1];    "A3,B3,C3,D3
```

TL/L/9991-25

FIGURE 3.4. ABEL Input File

3.0 Quad 4-to-1 Multiplexer (Continued)

```
ABEL(tm) Version 1.19  -  Document Generator
Quad 4 to 1 Multiplexer in a GAL20V8        April 17, 1986
National Semiconductor                             Joe Eng
Equations for Module quad_4to1_mux

Device U8


    Reduced Equations:

      Aout = (A0 & !S0 & !S1
              # A1 & S0 & !S1
              # A2 & !S0 & S1
              # A3 & S0 & S1);


      Bout = (B0 & !S0 & !S1
              # B1 & S0 & !S1
              # B2 & !S0 & S1
              # B3 & S0 & S1);


      Cout = (C0 & !S0 & !S1
              # C1 & S0 & !S1
              # C2 & !S0 & S1
              # C3 & S0 & S1);


      Dout = (D0 & !S0 & !S1
              # D1 & S0 & !S1
              # D2 & !S0 & S1
              # D3 & S0 & S1);
```

TL/L/9991–26

FIGURE 3.5. Reduced ABEL Equations

3.0 Quad 4-to-1 Multiplexer (Continued)

```
QP24* QF2706*
L0000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
1101111011101111111111111111111111111111
0111111011011111111111111111111111111111
1111010111101111111111111111111111111111
1111110101011111111111111111111111111111
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
1111111011100111111111111111111111111111
1111111011011111011111111111111111111111
1111110111101111111101111111111111111111
1111110111011111111111110111111111111111
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
1111111011101111111111111111011111111111
1111111011011111111111111111111101111111
1111110111101111111111111111111111110111
1111110111011111111111111111111111111101
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
1111111011101111111111111111111111011111
1111111011011111111111111111110111111111
1111110111101111111111111101111111111111
1111110111011111111111011111111111111111
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000000000000000000000*
L2560
01111000*
L2568
0000000000000000000000000000000000000000000000000000000000000000*
L2632
1*
L2633
0*
L2634
0*
L2635
0*
L2636
0*
L2637
1*
L2638
1*
L2639
1*
L2640
1111111111111111111111111111111111111111111111111111111111111111*
L2704
10*
V0001 10000100001N00001LLLH00N*
V0002 10000100001N00001LLHL10N*
V0003 10000100001N00001LHLL01N*
V0004 10000100001N00001HLLL11N*
V0005 11101101101N10111LHHH00N*
V0006 11101101101N10111HLHH10N*
V0007 11101101101N10111HHLH01N*
V0008 11101101101N10111HHHL11N*
C5127*
```

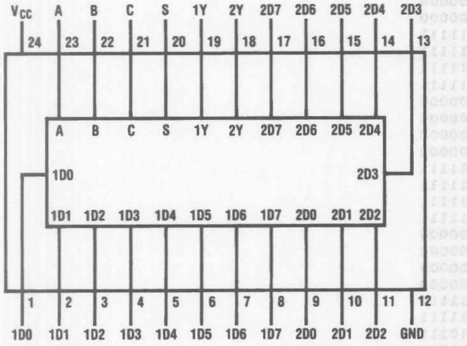
TL/L/9991-27

FIGURE 3.6. 'Fuse' Map

4.0 Dual 8-to-1 Multiplexer

The Dual 8:1 Mux selects one of eight inputs, D0 through D7, specified by three binary select inputs, A, B and C. The true data is output on Y when strobed by S. The circuit is implemented using a GAL22V10.

LOGIC SYMBOL


V_{CC} A B C S 1Y 2Y 2D7 2D6 2D5 2D4 2D3
24 23 22 21 20 19 18 17 16 15 14 13
A B C S 1Y 2Y 2D7 2D6 2D5 2D4
1D0
2D3
1D1 1D2 1D3 1D4 1D5 1D6 1D7 2D0 2D1 2D2
1 2 3 4 5 6 7 8 9 10 11 12
1D0 1D1 1D2 1D3 1D4 1D5 1D6 1D7 2D0 2D1 2D2 GND

TL/L/9991-28

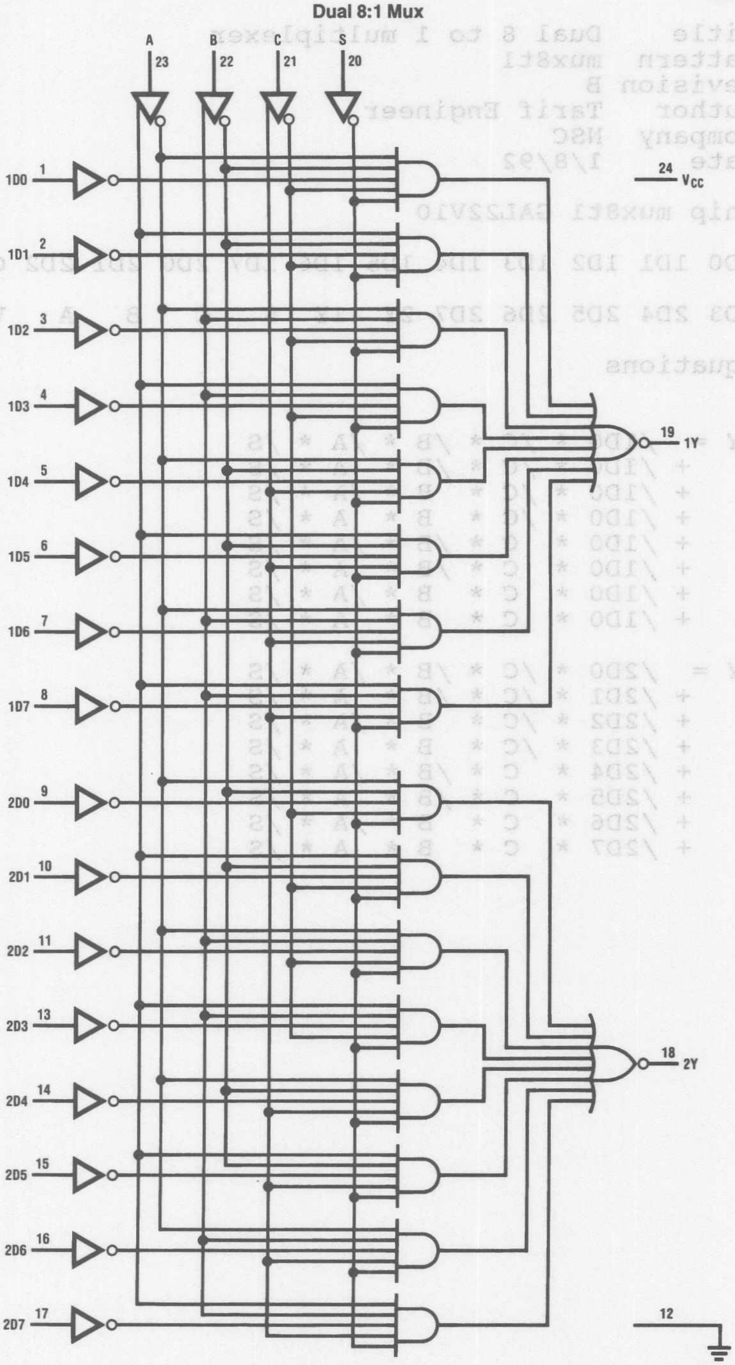
Pinout

FUNCTION TABLE

Inputs				Output
Select			Strobe	Y
C	B	A	S	
X	X	X	H	H
L	L	L	L	D0
L	L	H	L	D1
L	H	L	L	D2
L	H	H	L	D3
H	L	L	L	D4
H	L	H	L	D5
H	H	L	L	D6
H	H	H	L	D7

4.0 Dual 8-to-1 Multiplexer (Continued)

LOGIC DIAGRAM



TL/L/9991-29

```
title    Dual 8 to 1 multiplexer
pattern  mux8t1
revision B
author   Tarif Engineer
company  NSC
Date     1/8/92

chip mux8t1 GAL22V10

1D0 1D1 1D2 1D3 1D4 1D5 1D6 1D7 2D0 2D1 2D2 GND

2D3 2D4 2D5 2D6 2D7 2Y  1Y  S   C   B   A   VCC

equations


/1Y =  /1D0 * /C * /B * /A * /S
     + /1D0 * /C * /B *  A * /S
     + /1D0 * /C *  B * /A * /S
     + /1D0 * /C *  B *  A * /S
     + /1D0 *  C * /B * /A * /S
     + /1D0 *  C * /B *  A * /S
     + /1D0 *  C *  B * /A * /S
     + /1D0 *  C *  B *  A * /S

/2Y =  /2D0 * /C * /B * /A * /S
     + /2D1 * /C * /B *  A * /S
     + /2D2 * /C *  B * /A * /S
     + /2D3 * /C *  B *  A * /S
     + /2D4 *  C * /B * /A * /S
     + /2D5 *  C * /B *  A * /S
     + /2D6 *  C *  B * /A * /S
     + /2D7 *  C *  B *  A * /S
```

TL/L/9991-F0

4.0 Dual 8-to-1 Multiplexer (Continued)

OPAL™ JEDEC FILE

```
☻
GAL22V10
EQN2JED - Boolean Equations to JEDEC file assembler (Version V029)
Copyright (c) National Semiconductor Corporation 1990,1991
Assembled from "C:\OPAL102\MUX8T1.EQN". Date: 1-9-92
         title    Dual 8 to 1 multiplexer
         pattern  mux8t1
         revision B
         author   Tarif Engineer
         company  NSC
         Date     1/8/92

         *
NOTE PINS 1D0:1 1D1:2 1D2:3 1D3:4 1D4:5 1D5:6 1D6:7 1D7:8 2D0:9*
NOTE PINS 2D1:10 2D2:11 GND:12 2D3:13 2D4:14 2D5:15 2D6:16 2D7:17*
NOTE PINS 2Y:18 1Y:19 S:20 C:21 B:22 A:23 VCC:24*
QF5892*QP24*F0*
L2156
11111111111111111111111111111111111111111111
10101110111011101111111111111111111111111111
10011110111011101111111111111111111111111111
10101101111011101111111111111111111111111111
10011101111011101111111111111111111111111111
10101110110111101111111111111111111111111111
10011110110111101111111111111111111111111111
10101101110111101111111111111111111111111111
10011101110111101111111111111111111111111111*
L2904
11111111111111111111111111111111111111111111
11101110111011101111111111111111101111111111
11011110111011101111111111111111111110111111
11101101111011101111111111111111111111111011
11011101111011101111111111111111111111111110
11101110110111101111111111111111111111101111
11011110110111101111111111111111111011111111
11101101110111101111111111111110111111111111
11011101110111101111111111101111111111111111*
L5808
01010101010101010101*
L5828
0000000000000000000000000000000000000000000000000000000000000000*
C551B*
♥0000
```

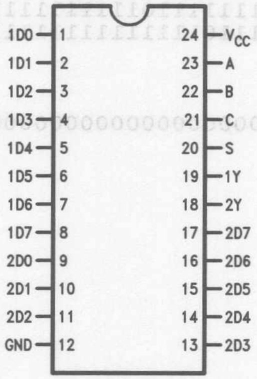
TL/L/9991-F1

4.0 Dual 8-to-1 Multiplexer (Continued)

```
EQN2JED - Boolean Equations to JEDEC file assembler (Version V029)
Copyright (c) National Semiconductor Corporation 1990,1991

Document file for C:\OPAL102\MUX8T1.EQN
Device: G22V10

$LABELS 24 1D0 1D1 1D2 1D3 1D4 1D5 1D6 1D7 2D0 2D1 2D2 GND 2D3 2D4 2D5 2D6
 2D7 2Y 1Y S C B A VCC

Pin    Label               Type
---    -----               ----
1      1D0                 pos,com input
2      1D1                 unused
3      1D2                 unused
4      1D3                 unused
5      1D4                 unused
6      1D5                 unused
7      1D6                 unused
8      1D7                 unused
9      2D0                 pos,com input
10     2D1                 pos,com input
11     2D2                 pos,com input
12     GND                 ground pin
13     2D3                 pos,com input
14     2D4                 pos,com input
15     2D5                 pos,com input
16     2D6                 pos,com input
17     2D7                 pos,com input
18     2Y                  neg,trst,com output
19     1Y                  neg,trst,com output
20     S                   pos,com input
21     C                   pos,com input
22     B                   pos,com input
23     A                   pos,com input
24     VCC                 power pin
```

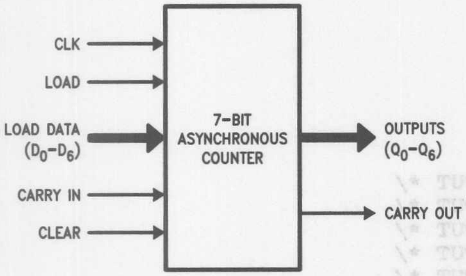
TL/L/9991-F4

Chip Diagram (DIP)



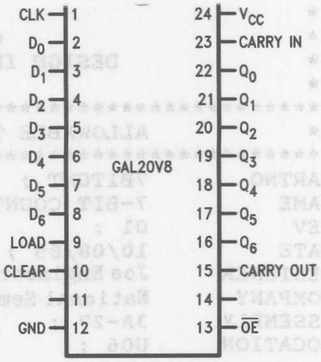
TL/L/9991-53

example, a GAL20V8 implements a seven bit counter with asynchronous carry-out and load functions. As illustrated in the block diagram *(Figure 5.1)* and pinout diagram *(Figure 5.2)*, the carry-in and carry-out pins make the counter fully cascadable to form larger counters. The CUPL design input files are shown in *Figure 5.3*, and simulation files in *Figure 5.4*. Note that the counter requires seven registers and one asynchronous output, taking full advantage of the generic architecture of the GAL20V8.



TL/L/9991-37

FIGURE 5.1. Block Diagram



TL/L/9991-38

FIGURE 5.2. Pinout Diagram

```
/*                     CUPL INPUT FILE                              */
/*             DESIGN INPUT FOR 7-BIT COUNTER                       */
/*                                                                  */
/********************************************************************/
/*          ALLOWABLE TARGET DEVICE:   GAL20V8                      */
/********************************************************************/
PARTNO         7BITCNT ;
NAME           7-BIT COUNTER ;
REV            01 ;
DATE           10/08/85 ;
DESIGNER       Joe Engineer;
COMPANY        National Semiconductor;
ASSEMBLY       3A-27 ;
LOCATION       U06 ;

PIN 1 = CLK ;               /* CLOCK INPUT */
PIN 2 = D0 ;                /* DATA0 INPUT */
PIN 3 = D1 ;                /* DATA1 INPUT */
PIN 4 = D2 ;                /* DATA2 INPUT */
PIN 5 = D3 ;                /* DATA3 INPUT */
PIN 6 = D4 ;                /* DATA4 INPUT */
PIN 7 = D5 ;                /* DATA5 INPUT */
PIN 8 = D6 ;                /* DATA6 INPUT */
PIN 9 = LD ;                /* LOAD CONTROL */
PIN 10 = CLEAR;             /* ASYNCHRONOUS CARRY-IN */

PIN 13 = !OE ;              /* OUTPUT ENABLE */
PIN 15 = CARRYOUT ;
PIN 16 = Q6 ;               /* COUNTER MSB */
PIN 17 = Q5 ;
PIN 18 = Q4 ;
PIN 19 = Q3 ;
PIN 20 = Q2 ;
PIN 21 = Q1 ;
PIN 22 = Q0 ;               /* COUNTER LSB */
PIN 23 = CARRYIN ;          /* CARRY-IN FOR CASCADING */
```

TL/L/9991-39

FIGURE 5.3. CUPL Design Input File

5.0 7-Bit Counter with Parallel Load (Continued)

```
Q0.D = (LD   & D0                                  /* LOAD D0 */
     # !LD & !Q0 & CARRYIN) & !CLEAR;              /*  TOGGLE */

Q1.D = (LD   & D1                                  /* LOAD D1 */
     # !LD& !Q1 &  Q0 & CARRYIN                    /*  TOGGLE */
     # !LD&  Q1 & !Q0) & !CLEAR;                   /*   HOLD  */

Q2.D = (LD   & D2                                  /* LOAD D2 */
     # !LD& !Q2 &  Q1 & Q0 & CARRYIN               /*  TOGGLE */
     # !LD&  Q2 & !Q1                              /*   HOLD  */
     # !LD&  Q2 & !Q0) & !CLEAR;                   /*   HOLD  */

Q3.D = (LD   & D3                                  /* LOAD D3 */
     # !LD& !Q3 &  Q2 & Q1 & Q0 & CARRYIN          /*  TOGGLE */
     # !LD&  Q3 & !Q2                              /*   HOLD  */
     # !LD&  Q3 & !Q1                              /*   HOLD  */
     # !LD&  Q3 & !Q0) & !CLEAR;                   /*   HOLD  */

Q4.D = (LD   & D4                                  /* LOAD D4 */
     # !LD& !Q4& Q3 & Q2 & Q1 & Q0 & CARRYIN       /* TOGGLE */
     # !LD& Q4 & !Q3                               /*  HOLD   */
     # !LD& Q4 & !Q2                               /*  HOLD   */
     # !LD& Q4 & !Q1                               /*  HOLD   */
     # !LD& Q4 & !Q0) & !CLEAR;                    /*  HOLD   */

Q5.D = (LD   & D5                                  /* LOAD D5 */
     # !LD& !Q5& Q4 & Q3 & Q2 & Q1 & Q0
       & CARRYIN                                   /*  TOGGLE */
     # !LD& Q5 & !Q4                               /*   HOLD  */
     # !LD& Q5 & !Q3                               /*   HOLD  */
     # !LD& Q5 & !Q2                               /*   HOLD  */
     # !LD& Q5 & !Q1                               /*   HOLD  */
     # !LD& Q5 & !Q0) & !CLEAR;                    /*   HOLD  */

Q6.D = (LD   & D6                                  /* LOAD D6 */
     # !LD& !Q6& Q5 & Q4 & Q3 & Q2 & Q1 & Q0
       & CARRYIN                                   /*  TOGGLE */
     # !LD& Q6 & !Q5                               /*   HOLD  */
     # !LD& Q6 & !Q4                               /*   HOLD  */
     # !LD& Q6 & !Q3                               /*   HOLD  */
     # !LD& Q6 & !Q2                               /*   HOLD  */
     # !LD& Q6 & !Q1                               /*   HOLD  */
     # !LD& Q6 & !Q0) & !CLEAR;                    /*   HOLD  */

CARRYOUT = !LD & Q6 & Q5 & Q4 & Q3 & Q2 & Q1 & Q0
           & CARRYIN;                              /* CARRY-OUT */
```

TL/L/9991-40

FIGURE 5.3. CUPL Design Input File (Continued)

5.0 7-Bit Counter with Parallel Load (Continued)

```
/******************************************************************/
/*                                                                */
/*                        CUPL INPUT FILE                         */
/*                  SIMULATION FOR 7-BIT COUNTER                  */
/*                                                                */
/******************************************************************/
/*          ALLOWABLE TARGET DEVICE: GAL20V8                      */
/******************************************************************/
PARTNO      7BITCNT ;
NAME        7-BIT COUNTER ;
REV         01 ;
DATE        10/08/85 ;
DESIGNER    Joe Engineer;
COMPANY     National Semiconductor;
ASSEMBLY    3A-27 ;
LOCATION    U06 ;
ORDER:

  CLK, !OE, CLEAR, LD, CARRYIN, D6, D5, D4, D3, D2, D1, D0, Q6,
  Q5, Q4, Q3, Q2, Q1, Q0, CARRYOUT;
VECTORS:

$msg"                             C ";
$msg" C ! C   C                   O ";
$msg" L O L L I DDDDDDD QQQQQQQ U ";
$msg" K E R D N 6543210 6543210 T ";
$msg" --------------------------- ";
      0 1 X X X XXXXXXX ZZZZZZZ X  /* TEST HI-Z  */
      C 0 1 X X XXXXXXX LLLLLLL L /* TEST CLEAR */
      C 0 0 1 X 1111111 HHHHHHH L  /* LOAD ONES  */
      C 0 0 1 X 0000000 LLLLLLL L  /* LOAD ZEROS */
      C 0 0 0 1 XXXXXXX LLLLLLH L  /* COUNT=1 */
      C 0 0 0 1 XXXXXXX LLLLLHL L  /* COUNT=2 */
      C 0 0 0 1 XXXXXXX LLLLLHH L  /* COUNT=3 */
      C 0 0 0 1 XXXXXXX LLLLHLL L  /* COUNT=4 */
      C 0 0 0 1 XXXXXXX LLLLHLH L  /* COUNT=5 */
      C 0 0 0 1 XXXXXXX LLLLHHL L  /* COUNT=6 */
      C 0 0 0 1 XXXXXXX LLLLHHH L  /* COUNT=7 */
      C 0 0 0 1 XXXXXXX LLLHLLL L /* COUNT=8 */
      C 0 0 0 1 XXXXXXX LLLHLLH L  /* COUNT=9 */
      C 0 0 0 1 XXXXXXX LLLHLHL L  /* COUNT=10 */
      C 0 0 0 1 XXXXXXX LLLHLHH L  /* COUNT=11 */
      C 0 0 0 1 XXXXXXX LLLHHLL L  /* COUNT=12 */
      C 0 0 0 1 XXXXXXX LLLHHLH L  /* COUNT=13 */
      C 0 0 0 1 XXXXXXX LLLHHHL L  /* COUNT=14 */
      C 0 0 0 1 XXXXXXX LLLHHHH L  /* COUNT=15 */
      C 0 0 0 1 XXXXXXX LLHLLLL L  /* COUNT=16 */
      C 0 0 0 1 XXXXXXX LLHLLLH L  /* COUNT=17 */
      C 0 0 0 1 XXXXXXX LLHLLHL L  /* COUNT=18 */
      C 0 0 0 1 XXXXXXX LLHLLHH L  /* COUNT=19 */
      C 0 0 0 1 XXXXXXX LLHLHLL L  /* COUNT=20 */
      C 0 0 0 1 XXXXXXX LLHLHLH L  /* COUNT=21 */
      C 0 0 0 1 XXXXXXX LLHLHHL L  /* COUNT=22 */
      C 0 0 0 1 XXXXXXX LLHLHHH L  /* COUNT=23 */
      C 0 0 0 1 XXXXXXX LLHHLLL L  /* COUNT=24 */
      C 0 0 0 1 XXXXXXX LLHHLLH L  /* COUNT=25 */
      C 0 0 0 1 XXXXXXX LLHHLHL L  /* COUNT=26 */
      C 0 0 0 1 XXXXXXX LLHHLHH L  /* COUNT=27 */
      C 0 0 0 1 XXXXXXX LLHHHLL L  /* COUNT=28 */
      C 0 0 0 1 XXXXXXX LLHHHLH L  /* COUNT=29 */
      C 0 0 0 1 XXXXXXX LLHHHHL L  /* COUNT=30 */
      C 0 0 0 1 XXXXXXX LLHHHHH L  /* COUNT=31 */
      C 0 0 0 1 XXXXXXX LHLLLLL L  /* COUNT=32 */
      C 0 0 0 1 XXXXXXX LHLLLLH L  /* COUNT=33 */
      C 0 0 1 X 0111111 LHHHHHH L  /* LOAD=63 TO OBSERVE MSB TOGGLE */
      C 0 0 0 1 XXXXXXX HLLLLLL L  /* COUNT=64, OBSERVE MSB */
      C 0 0 0 1 XXXXXXX HLLLLLH L  /* COUNT=65, OBSERVE MSB */
      C 0 0 1 X 1111110 HHHHHHL L  /* LOAD=126 TO OBSERVE CARRY */
      C 0 0 0 1 XXXXXXX HHHHHHH H  /* COUNT=127, OBSERVE CARRY  */
      C 0 0 0 1 XXXXXXX LLLLLLL L  /* COUNT=0, OBSERVE CARRY    */
```

TL/L/9991-50

FIGURE 5.4. CUPL Simulation File

6.0 10-Bit Up/Down Counter

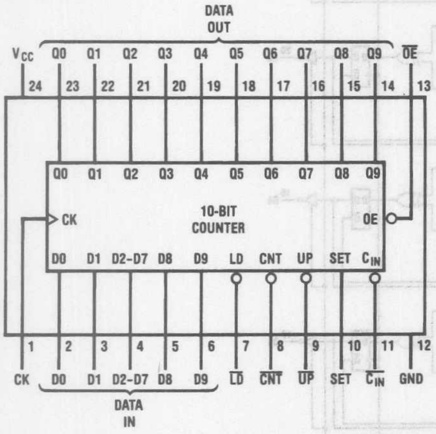
The ten-bit up/down counter can count up, count down, set all output to high, disable the output (high impedance), and load 2 LSB's, 2 MSB's and 6 middle bits high or low as a group. All operations are synchronous with the rising edge of the clock. SET overrides LOAD, COUNT and HOLD. LOAD overrides COUNT and HOLD.

C_{IN} enable counting operation or hold it. COUNT Up or Down depend on $\overline{UP}$ signal.

All outputs are enabled when $\overline{OE}$ is low, otherwise HIGH-Z.

This circuit is implemented using a PAL20X10, with the exclusive-or function the PAL20X10 facilitates design of counter and state sequences with minimum propagation delay. The PAL20X10 offers an efficient means of implementing counters. Normal PAL & PLA implementation would require addition terms for the XOR functions. Having 10 output the PAL20X10 supersets 20 and 24 medium PAL's in this specific application. On power up all registers are reset to simplify sequential circuit design.

LOGIC SYMBOL



TL/L/9991-51

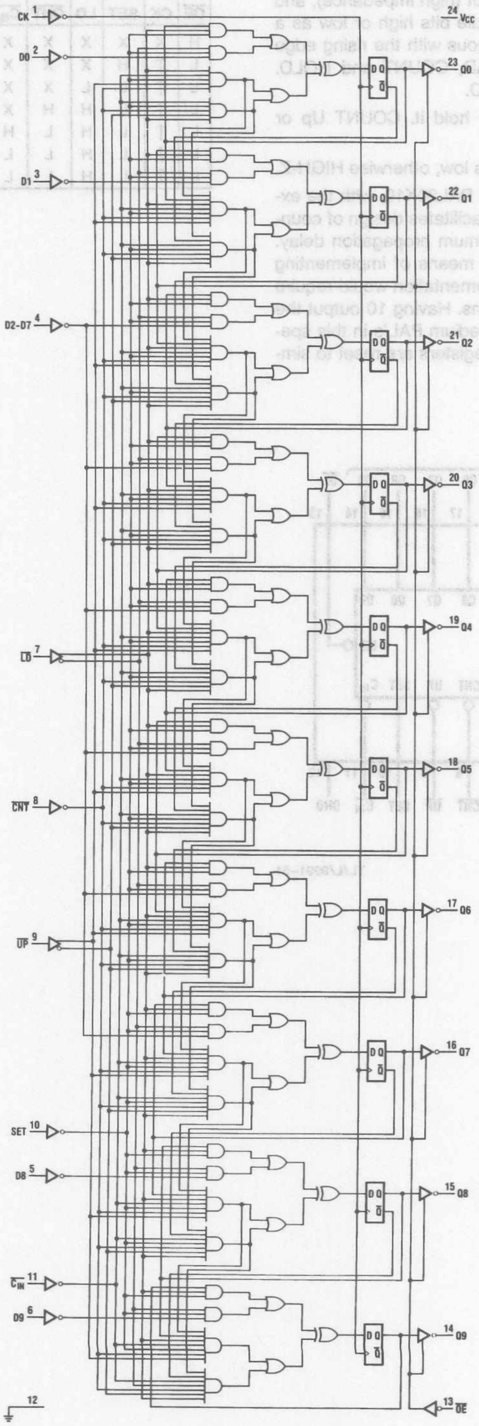
Pinout

FUNCTION TABLE

$\overline{OE}$	CK	SET	LD	$\overline{CNT}$	$\overline{C_{IN}}$	$\overline{UP}$	D9-D0	Q9-Q0	Operation
H	X	X	X	X	X	X	X	Z	Hi-Z
L	↑	H	X	X	X	X	X	H	Set all HIGH
L	↑	L	L	X	X	X	D	D	LOAD D
L	↑	L	H	H	X	X	X	Q	HOLD
L	↑	L	H	L	H	X	X	Q	HOLD
L	↑	L	H	L	L	L	X	Q plus 1	Count UP
L	↑	L	H	L	L	H	X	Q minus 1	Count DN

6.0 10-Bit Up/Down Counter (Continued)

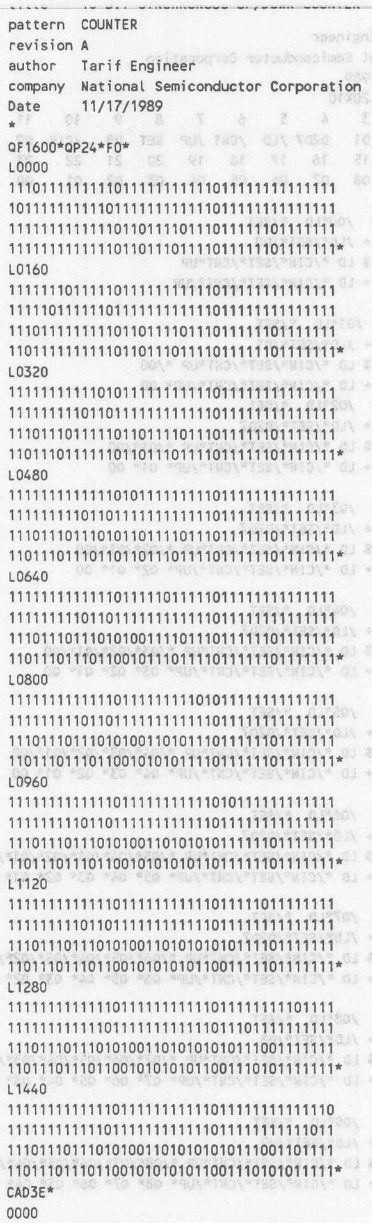
LOGIC DIAGRAM



TL/L/9991-52

```
pattern  COUNTER
revision A
author   Tarif Engineer
company  National Semiconductor Corporation
Date     11/17/1989
chip COUNTER PAL20X10
; pin 1    2    3    4    5    6    7    8    9    10   11   12
      CLK  D0   D1   D2D7 /LD  /CNT /UP  SET  D8   /CIN D9   GND
; pin 13   14   15   16   17   18   19   20   21   22   23   24
      /OE  Q9   Q8   Q7   Q6   Q5   Q4   Q3   Q2   Q1   Q0   VCC
equations
          /Q0 :=  /Q0*LD  */SET
                + /LD*/SET*/D0
                $ LD */CIN*/SET*/CNT*UP
                + LD */CIN*/SET*/CNT*/UP

         /Q1 :=  /Q1*LD  */SET
                + /LD*/SET*/D1
                $ LD */CIN*/SET*/CNT*UP */Q0
                + LD */CIN*/SET*/CNT*/UP* Q0
          /Q2 :=  /Q2*LD  */SET
                + /LD*/SET*/D2D7
                $ LD */CIN*/SET*/CNT*UP */Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q1* Q0

          /Q3 :=  /Q3*LD  */SET
                + /LD*/SET*/D2D7
                $ LD */CIN*/SET*/CNT*UP */Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q2* Q1* Q0

          /Q4 :=  /Q4*LD  */SET
                + /LD*/SET*/D2D7
                $ LD */CIN*/SET*/CNT*UP */Q3*/Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q3* Q2* Q1* Q0

          /Q5 :=  /Q5*LD  */SET
                + /LD*/SET*/D2D7
                $ LD */CIN*/SET*/CNT*UP */Q4*/Q3*/Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q4* Q3* Q2* Q1* Q0

          /Q6 :=  /Q6*LD  */SET
                + /LD*/SET*/D2D7
                $ LD */CIN*/SET*/CNT*UP */Q5*/Q4*/Q3*/Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q5* Q4* Q3* Q2* Q1* Q0

          /Q7 :=  /Q7*LD  */SET
                + /LD*/SET*/D2D7
                $ LD */CIN*/SET*/CNT*UP */Q6*/Q5*/Q4*/Q3*/Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q6* Q5* Q4* Q3* Q2* Q1* Q0

          /Q8 :=  /Q8*LD  */SET
                + /LD*/SET*/D8
                $ LD */CIN*/SET*/CNT*UP */Q7*/Q6*/Q5*/Q4*/Q3*/Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q7* Q6* Q5* Q4* Q3* Q2* Q1* Q0

          /Q9 :=  /Q9*LD  */SET
                + /LD*/SET*/D9
                $ LD */CIN*/SET*/CNT*UP */Q8*/Q7*/Q6*/Q5*/Q4*/Q3*/Q2*/Q1*/Q0
                + LD */CIN*/SET*/CNT*/UP* Q8* Q7* Q6* Q5* Q4* Q3* Q2* Q1* Q0
```

TL/L/9991-F6



```
title    10 BIT SYNCHRONOUS UP/DOWN COUNTER
pattern  COUNTER
revision A
author   Tarif Engineer
company  National Semiconductor Corporation
Date     11/17/1989
*
QF1600*QP24*F0*
L0000
1110111111111011111111111011111111111111
1011111111110111111111111011111111111111
1111111111111011011110111011111101111111
1111111111111011011101111011111101111111*
L0160
1111111011111011111111111011111111111111
1111101111110111111111111011111111111111
1110111111111011011110111011111101111111
1101111111111011011101111011111101111111*
L0320
1111111111101011111111111011111111111111
1111111110110111111111111011111111111111
1110111011111011011110111011111101111111
1101110111111011011101111011111101111111*
L0480
1111111111111010111111111011111111111111
1111111110110111111111111011111111111111
1110111011101011011110111011111101111111
1101110111011011011101111011111101111111*
L0640
1111111111111011111011111011111111111111
1111111110110111111111111011111111111111
1110111011101010011110111011111101111111
1101110111011001011101111011111101111111*
L0800
1111111111111011111111101011111111111111
1111111110110111111111111011111111111111
1110111011101010011010111011111101111111
1101110111011001010101111011111101111111*
L0960
1111111111111011111111111010111111111111
1111111110110111111111111011111111111111
1110111011101010011010101011111101111111
1101110111011001010101011011111101111111*
L1120
1111111111111011111111111011111011111111
1111111110110111111111111011111111111111
1110111011101010011010101010111101111111
1101110111011001010101011001111101111111*
L1280
1111111111111011111111111011111111101111
1111111111110111111111111011101111111111
1110111011101010011010101010111001111111
1101110111011001010101011001110101111111*
L1440
1111111111111011111111111011111111111110
1111111111110111111111111011111111111011
1110111011101010011010101010111001101111
1101110111011001010101011001110101011111*
CAD3E*
0000
```

TL/L/9991-F7

6.0 10-Bit Up/Down Counter (Continued)

The same design can be implemented in a GAL22V10 by converting the Exclusive OR function to OR & AND functions which can be done manually by applying DeMorgan's Low (see appendix A) to the Exclusive OR function

F = A $ B " Xor relations "

F = (A & /B) + (/A & B) " OR, AND relation "

Or the conversion can be done in an easier way by using OPAL feature. The software will automatically expand and minimize the equations. To get more information on the conversion procedure refer to PAL to GAL conversion note in this data book (APP Note # 799).

The following shows the final equations after converting and minimizing the previous design.

```
chip 10COUNT GAL22V10

CLK LD ST UP CNT CIN D9 D8 D2D7 D1 D0 GND
nc  Q0 Q3 Q2 Q8  Q9  Q5 Q7 Q6   Q4 Q1 VCC

equations

Q0 := LD * /ST * /CNT * /CIN * /Q0
    + LD * /ST * CIN * Q0
    + LD * /ST * CNT * Q0
    + /LD * /ST * /D0
Q1 := /Q1 * LD * /ST * UP * /CNT * /CIN * /Q0
    + /Q1 * LD * /ST * /UP * /CNT * /CIN * Q0
    + Q1 * LD * /ST * /UP * /Q0
    + Q1 * LD * /ST * UP * Q0
    + /LD * /ST * /D1
    + Q1 * LD * /ST * CIN
    + Q1 * LD * /ST * CNT
Q2 := Q1 * LD * /ST * Q2 * /Q0
    + LD * /ST * UP * Q2 * Q0
    + /Q1 * LD * /ST * /UP * Q2
    + /Q1 * LD * /ST * UP * /CNT * /CIN * /Q2 * /Q0
    + Q1 * LD * /ST * /UP * /CNT * /CIN * /Q2 * Q0
    + /LD * /ST * /D2D7
    +LD * /ST * CIN * Q2
    + LD * /ST * CNT * Q2
Q3 := Q1 * LD * /ST * Q3 * /Q0
    + LD * /ST * /Q2 * Q3 * Q0
    + /Q1 * LD * /ST * /UP * Q3
    + LD * /ST * UP * Q2 * Q3
    + /Q1 * LD * /ST * UP * /CNT * /CIN * /Q2 * /Q3 * /Q0
    + Q1 * LD * /ST * /UP * /CNT * /CIN * Q2 * /Q3 * Q0
    + /LD * /ST * /D2D7
    + LD * /ST * CIN * Q3
    + LD * /ST * CNT * Q3
Q4 := LD * Q4 * /ST * Q2 * /Q0
    + /Q1 * LD * Q4 * /ST * Q0
    + Q1 * LD * Q4 * /ST * UP
    + LD * Q4 * /ST * /Q2 * Q3
    + LD * Q4 * /ST * /UP * /Q3
    + /Q1 * LD * /Q4 * /ST * UP * /CNT * /CIN * /Q2 * /Q3 * /Q0
    + Q1 * LD * /Q4 * /ST * /UP * /CNT * /CIN * Q2 * Q3 * Q0
    + /LD * /ST * /D2D7
    + LD * Q4 * /ST * CIN
    + LD * Q4 * /ST * CNT
Q5 := LD * Q4 * /ST * Q5 * /Q0
    + LD * /ST * Q5 * /Q3 * Q0
    + /Q1 * LD * /ST * /UP * Q5
    + Q1 * LD * /Q4 * /ST * Q5
    + LD * /ST * Q5 * /Q2 * Q3
    + LD * /ST * UP * Q5 * Q2
    + /Q1 * LD * /Q4 * /ST * UP * /CNT * /Q5 * /CIN * /Q2 * /Q3 * /Q0
    + Q1 * LD * Q4 * /ST * /UP * /CNT * /Q5 * /CIN * Q2 * Q3 * Q0
    + /LD * /ST * /D2D7
    + LD * /ST * Q5 * CIN
    + LD * /ST * CNT * Q5
Q6 := LD * /ST * Q6 * Q2 * /Q0
    + LD * /ST * Q6 * /Q5 * Q0
    + /Q1 * LD * /ST * Q6 * Q5
    + Q1 * LD * /ST * Q6 * UP
    + LD * /ST * Q6 * /Q2 * Q3
    + LD * Q4 * /ST * Q6 * /Q3
    + LD * /Q4 * /ST * Q6 * /UP
    + /Q1 * LD * /Q4 * /ST * /Q6 * UP * /CNT * /Q5 * /CIN * /Q2 * /Q3
```

TL/L/9991-13

6.0 10-Bit Up/Down Counter (Continued)

```
           * /Q0
         + Q1 * LD * Q4 * /ST * /Q6 * /UP * /CNT * Q5 * /CIN * Q2 * Q3 * Q0
         + /LD * /ST * /D2D7
         + LD * /ST * Q6 * CIN
         + LD * /ST * Q6 * CNT
   Q7 := Q1 * LD * /ST * Q7 * /Q0
         + LD * /ST * Q7 * /Q2 * Q0
         + /Q1 * LD * /ST * /UP * Q7
         + LD * /ST * /Q6 * Q7 * Q2
         + LD * /ST * Q6 * Q7 * /Q3
         + LD * /ST * Q7 * /Q5 * Q3
         + LD * /Q4 * /ST * Q7 * Q5
         + LD * Q4 * /ST * UP * Q7
         + /Q1 * LD * /Q4 * /ST * /Q6 * UP * /Q7 * /CNT * /Q5 * /CIN * /Q2
           * /Q3 * /Q0
         + Q1 * LD * Q4 * /ST * Q6 * /UP * /Q7 * /CNT * Q5 * /CIN * Q2 * Q3
           * Q0
         + /LD * /ST * /D2D7
         + LD * /ST * Q7 * CIN
         + LD * /ST * Q7 * CNT
   Q8 := LD * /ST * Q8 * Q2 * /Q0
         + LD * /ST * /Q7 * Q8 * Q0
         + /Q1 * LD * /ST * Q8 * Q3
         + Q1 * LD * /ST * /Q5 * Q8
         + LD * Q4 * /ST * Q8 * /Q2
         + LD * /ST * Q7 * Q8 * /Q3
         + LD * /Q4 * /ST * Q6 * Q8
         + LD * /ST * UP * Q5 * Q8
         + LD * /ST * /Q6 * /UP * Q8
         + /Q1 * LD * /Q4 * /ST * /Q6 * UP * /Q7 * /CNT * /Q5 * /CIN * /Q8
           * /Q2 * /Q3 * /Q0
         + Q1 * LD * Q4 * /ST * Q6 * /UP * Q7 * /CNT * Q5 * /CIN * /Q8 * Q2
           * Q3 * Q0
         + /LD * /ST * /D8
         + LD * /ST * CIN * Q8
         + LD * /ST * CNT * Q8
   Q9 := LD * /ST * Q9 * Q3 * /Q0
         + LD * /Q4 * /ST * Q9 * Q0
         + /Q1 * LD * Q4 * /ST * Q9
         + Q1 * LD * /ST * Q9 * /Q8
         + LD * /ST * Q9 * Q8 * /Q2
         + LD * /ST * UP * Q9 * Q2
         + LD * /ST * Q7 * Q9 * /Q3
         + LD * /ST * Q6 * /Q5 * Q9
         + LD * /ST * /Q7 * Q5 * Q9
         + LD * /ST * /Q6 * /UP * Q9
         + /Q1 * LD * /Q4 * /ST * /Q6 * UP * /Q7 * /CNT * /Q5 * /CIN * /Q9
           * /Q8 * /Q2 * /Q3 * /Q0
         + Q1 * LD * Q4 * /ST * Q6 * /UP * Q7 * /CNT * Q5 * /CIN * /Q9 * Q8
           * Q2 * Q3 * Q0
         + /LD * /ST * /D9
         + LD * /ST * CIN * Q9
         + LD * /ST * CNT * Q9
```

TL/L/9991-13

6.0 10-Bit Up/Down Counter (Continued)

```
GAL22V10
*
QF5892*QP24*F0*
L0044
11111111111111111111111111111111111111111111
11010111101101111011101111111111111111011111
11010111101110111011101111111111111111101111
11100111101110111111111111111111111111011111
11100111101101111111111111111111111111101111
11111011101111111111111111111111111110111111
11100111101111111111011111111111111111111111
11100111101111110111111111111111111111111111*
L0440
11111111111111111111111111111111111111111111
11110110101111111111111111111110111111011111
11010110101111111111111111111111111111101111
11100110101101111111111111111111111111111111
11110110101111111111111111111101111011111111
11110110101110111111111111111111110111111111
11010101101101111011101111111101110111011111
11100101101110111011101111111110111011101111
11111011101111111111111111111111101111111111
11110110101111111111011111111111111111111111
11110110101111110111111111111111111111111111*
L0924
11111111111111111111111111111111111111111111
11110111101011111111111111111110111111011111
11110111101011111101111111111111111111101111
11010111101011111110111111111111111111111111
11100111101001111111111111111111111111111111
11110111101011111111111111111101111011111111
11110110101011111111111111111111110111111111
11110101101010111111111111111111111111111111
11010101100101111001101111111101110111011111
11100110100110111010101111111110111011101111
11111011101111111111111111111111101111111111
11110111101011111111011111111111111111111111
11110111101011110111111111111111111111111111*
L1496
11111111111111111111111111111111111111111111
11100111101111101111111111111111111111011111
11110111101111101111111111111101111111101111
11010111101110101111111111111111111111111111
11110111100111101111111111111110111111111111
11110111101011101111111111111111110111111111
11110111101111101101111111111111111011111111
11110101101111101110111111111111111111111111
11110110101101101111111111111111111111111111
11010101100101011001101111111101110111011111
11100110101010011010101111111110111011101111
11111011101111111111111111111111101111111111
11110111101111101111011111111111111111111111
11110111101111100111111111111111111111111111*
L2156
11111111111111111111111111111111111111111111
11110110101111111110111111111111111111011111
11110111101111111110111111111111110111101111
11010111101110111110111111111111111111111111
11100101101111111110111111111111111111111111
11110111101111111110111111111101111011111111
11110111101101111110111111111110111111111111
11010101101101111001101111111101110111011111
11100110101110111001101111111110111011101111
11111011101111111111111111111111101111111111
11110111101111111110011111111111111111111111
11110111101111110110111111111111111111111111*
```

TL/L/9991-18

```
11111111111111111111111111111111111111111111
11110111101111111111111011111111111011011111
11110101101111111111111011111111111111101111
11010110101111111111111011111111111111111111
11100111101111111111111011011111111111111111
11110111101111111111111011101101111111111111
11110111101101111111111011111110111111111111
11110111101111101111111011111111110111111111
11110111101011111101111011111111111111111111
11110111101111011110111011111111111111111111
11110111100110111111111011111111111111111111
11010101100101011001100111011101110111011111
11100110101010101010100111101110111011101111
11111011101111111111111110111111111111111111
11110111101111111111011011111111111111111111
11110111101111110111111011111111111111111111*
L3652
11111111111111111111111111111111111111111111
11110111101111111111111111101110111111011111
11110111101111011111111111101111111111101111
11010111101111111111111111101111110111111111
11100111101111111101111111101111111111111111
11110110101111111111111111101101111111111111
11110111101111101111111111101111110111111111
11110101101011111111111111101111111111111111
11110111101101111110111111101111111111111111
11110111100110111111111111101111111111111111
11010101100101011001101111011101110111011111
11100110101010101010101111011110111011101111
11111011101111111111111111111011111111111111
11110111101111111111011111101111111111111111
11110111101111110111111111101111111111111111*
L4312
11111111111111111111111111111111111111111111
11100111101111111111111111111110111111011111
11110111101101111111111111111110111111101111
11010111101110111111111111111110111111111111
11010111101101111011101111111101111111011111
11100111101110111011101111111101111111101111
11111011101111111111111111111111110111111111
11110111101111111111011111111110111111111111
11110111101111110111111111111110111111111111*
L4884
11111111111111111111111111111111111111111111
11100111101111111111111111111111111011011111
11110111101111111111111111111101111011101111
11010111101110111111111111111111111011111111
11110111101101111111111111111110111011111111
11010111101101111011101111111101110111011111
11100111101110111011101111111110110111101111
11111011101111111111111111111111110111111111
11110111101111111111011111111111111011111111
11110111101111110111111111111111111011111111*
L5368
11111111111111111111111111111111111111111111
11110111101111110110111111111111111111011111
11110111101111111111011111111111111111101111
11110111101111110111111111111111111111101111
11111011101111111111111111111111111111111011*
L5808
10101010101010101010*
L5828
0000000000000000000000000000000000000000000000000000000000000000*
C2220*
♥0000
```

TL/L/9991-21

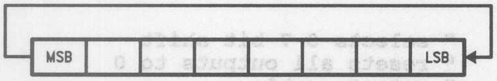
7.0 8-Bit Barrel Shifter

DESCRIPTION

The barrel shifter *(Figure 7.1)* is a specialized shift register that rotates data a selectable number of bit positions out of the most-significant bit and back into the least-significant bit—thus the name. Typical applications of a barrel shifter are floating-point arithmetic and display rotation on a graphics terminal.

Since our barrel shifter has 8 data inputs and 8 registered outputs, as well as control signals, the GAL20V8 is the PLD of choice. The shift-select inputs (S_0, S_1, S_2) determine the number of positions shifted, as described in the function table of *Figure 7.2*. The block diagram is shown in *Figure 7.3*, and the pinout in *Figure 7.4*. The clock (CLK) input gates input data synchronously to the output registers, and the output enable (OE) allows TRI-STATE® buffering of the Q outputs. The one remaining input is used for a reset (RS) function.

The ABEL design input files shown in *Figure 7.5* may appear tedious, but simply enumerate the eight different bit-shift possibilities for each output.

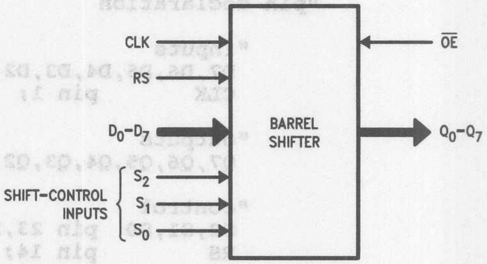


TL/L/9991-55

FIGURE 7.1. Barrel Shift Rotation

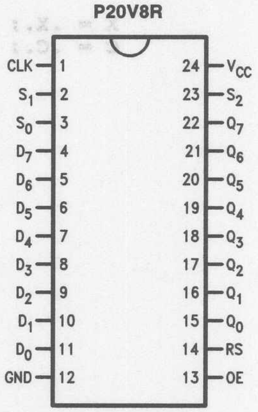
S_2	S_1	S_0	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
0	0	0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7
0	1	0	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6
0	1	1	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5
1	0	0	D_3	D_2	D_1	D_0	D_7	D_6	D_5	D_4
1	0	1	D_2	D_1	D_0	D_7	D_6	D_5	D_4	D_3
1	1	0	D_1	D_0	D_7	D_6	D_5	D_4	D_3	D_2
1	1	1	D_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1

FIGURE 7.2. Function Table



TL/L/9991-56

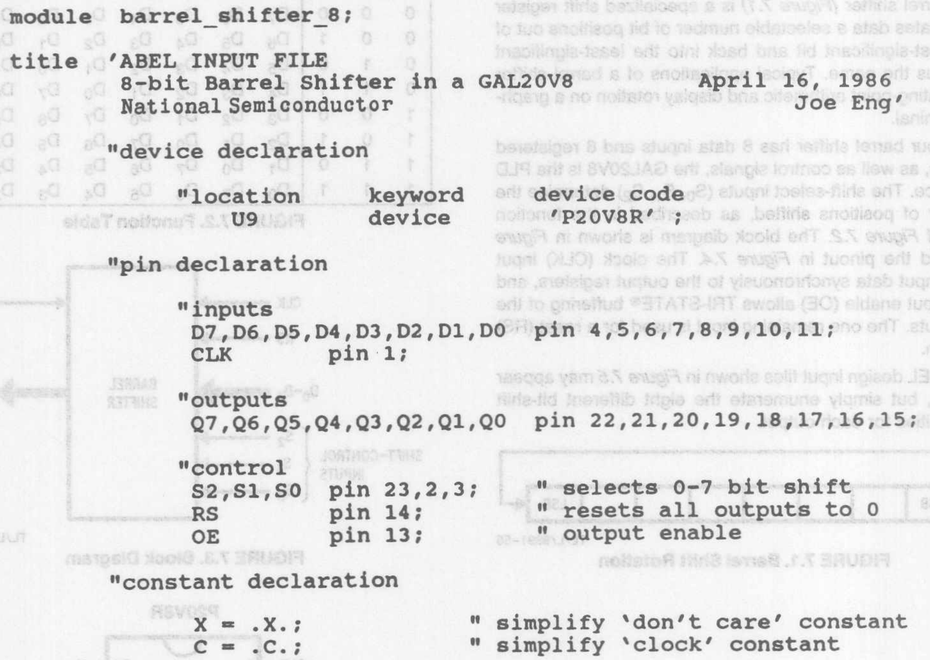
FIGURE 7.3. Block Diagram



TL/L/9991-57

FIGURE 7.4. Pinout Diagram

7.0 8-Bit Barrel Shifter (Continued)



```
module  barrel_shifter_8;

title  'ABEL INPUT FILE
        8-bit Barrel Shifter in a GAL20V8          April 16, 1986
        National Semiconductor                          Joe Eng'

       "device declaration

            "location     keyword    device code
                  U9      device     'P20V8R';

       "pin declaration

            "inputs
             D7,D6,D5,D4,D3,D2,D1,D0  pin 4,5,6,7,8,9,10,11;
             CLK        pin 1;

            "outputs
             Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0  pin 22,21,20,19,18,17,16,15;

            "control
             S2,S1,S0   pin 23,2,3;       " selects 0-7 bit shift
             RS         pin 14;           " resets all outputs to 0
             OE         pin 13;           " output enable

       "constant declaration

             X = .X.;                 " simplify 'don't care' constant
             C = .C.;                 " simplify 'clock' constant
```

TL/L/9991-45

FIGURE 7.5. ABEL Input File

```
                 ((!S2 & !S1 & !S0 & D0) #
                  (!S2 & !S1 & S0 & D7) #
                  (!S2 & S1 & !S0 & D6) #
                  (!S2 & S1 & S0 & D5) #
                  (S2 & !S1 & !S0 & D4) #
                  (S2 & !S1 & S0 & D3) #
                  (S2 & S1 & !S0 & D2) #
                  (S2 & S1 & S0 & D1));

Q1 := !RS & ((!S2 & !S1 & !S0 & D1) #
              (!S2 & !S1 & S0 & D0) #
              (!S2 & S1 & !S0 & D7) #
              (!S2 & S1 & S0 & D6) #
              (S2 & !S1 & !S0 & D5) #
              (S2 & !S1 & S0 & D4) #
              (S2 & S1 & !S0 & D3) #
              (S2 & S1 & S0 & D2));

Q2 := !RS & ((!S2 & !S1 & !S0 & D2) #
              (!S2 & !S1 & S0 & D1) #
              (!S2 & S1 & !S0 & D0) #
              (!S2 & S1 & S0 & D7) #
              (S2 & !S1 & !S0 & D6) #
              (S2 & !S1 & S0 & D5) #
              (S2 & S1 & !S0 & D4) #
              (S2 & S1 & S0 & D3));

Q3 := !RS & ((!S2 & !S1 & !S0 & D3) #
              (!S2 & !S1 & S0 & D2) #
              (!S2 & S1 & !S0 & D1) #
              (!S2 & S1 & S0 & D0) #
              (S2 & !S1 & !S0 & D7) #
              (S2 & !S1 & S0 & D6) #
              (S2 & S1 & !S0 & D5) #
              (S2 & S1 & S0 & D4));

Q4 := !RS & ((!S2 & !S1 & !S0 & D4) #
              (!S2 & !S1 & S0 & D3) #
              (!S2 & S1 & !S0 & D2) #
              (!S2 & S1 & S0 & D1) #
              (S2 & !S1 & !S0 & D0) #
              (S2 & !S1 & S0 & D7) #
              (S2 & S1 & !S0 & D6) #
              (S2 & S1 & S0 & D5));
```

TL/L/9991-46

FIGURE 7.5. ABEL Input File (Continued)

```
                     (!S2 & !S1 & S0 & D4) #
                    (!S2 & S1 & !S0 & D3) #
                     (!S2 & S1 & S0 & D2) #
                    (S2 & !S1 & !S0 & D1) #
                     (S2 & !S1 & S0 & D0) #
                     (S2 & S1 & !S0 & D7) #
                      (S2 & S1 & S0 & D6));

        Q6 := !RS & ((!S2 & !S1 & !S0 & D6) #
                     (!S2 & !S1 & S0 & D5) #
                    (!S2 & S1 & !S0 & D4) #
                     (!S2 & S1 & S0 & D3) #
                    (S2 & !S1 & !S0 & D2) #
                     (S2 & !S1 & S0 & D1) #
                     (S2 & S1 & !S0 & D0) #
                      (S2 & S1 & S0 & D7));

        Q7 := !RS & ((!S2 & !S1 & !S0 & D7) #
                     (!S2 & !S1 & S0 & D6) #
                    (!S2 & S1 & !S0 & D5) #
                     (!S2 & S1 & S0 & D4) #
                    (S2 & !S1 & !S0 & D3) #
                     (S2 & !S1 & S0 & D2) #
                     (S2 & S1 & !S0 & D1) #
                      (S2 & S1 & S0 & D0));

test_vectors  ([CLK,OE,RS,S2,S1,S0,D7..D0] -> [Q7..Q0])

   "  C
   "  L O R S   S D           D      Q             Q
   "  K E S 2 1 0 7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0
     [C,0,1,X,X,X,X,X,X,X,X,X,X,X] -> [0,0,0,0,0,0,0,0];  " set
     [C,0,0,0,0,0,0,0,0,1,1,1,1] -> [0,0,0,0,1,1,1,1];  " no shift
     [C,0,0,0,0,1,1,1,1,1,0,0,0,0] -> [1,1,1,0,0,0,0,1];  " shift 1
     [C,0,0,0,1,0,0,0,0,0,1,1,1,1] -> [0,0,1,1,1,1,0,0];  "       2
     [C,0,0,0,1,1,1,1,1,1,0,0,0,0] -> [1,0,0,0,0,1,1,1];  "       3
     [C,0,0,1,0,0,0,0,0,0,1,1,1,1] -> [1,1,1,1,0,0,0,0];  "       4
     [C,0,0,1,0,1,1,1,1,1,0,0,0,0] -> [0,0,0,1,1,1,1,0];  "       5
     [C,0,0,1,1,0,0,0,0,0,1,1,1,1] -> [1,1,0,0,0,0,1,1];  "       6
     [C,0,0,1,1,1,1,1,1,1,0,0,0,0] -> [0,1,1,1,1,0,0,0];  '       7

end barrel_shifter_8
```

TL/L/9991-47

FIGURE 7.5. ABEL Input File (Continued)

8.0 Hexadecimal 7-Segment Display Encoder

The increasing use of microcomputers has led to an increased need to display numbers in hexadecimal format (0–9, A–F). Standard drivers for this function are not available, so most applications are forced to use several packages to decode each digit of the display. Since 6 to 12 digits are often being displayed, this approach can become very expensive. This example demonstrates how the hexadecimal display format can be both decoded and the LED indicators driven using a single GAL for each digit of the display.

FUNCTIONAL DESCRIPTION

A hex decoder/lamp driver accepts a four-bit hex digit, converts it to its corresponding seven-segment display code, and activates the appropriate segments on the display. These drivers can be used in both direct-drive and multiplexed display applications. A single GAL can provide both the basic decode/drive functions, and additional useful features as well.

GENERAL DESCRIPTION

Figure 8.1 shows three digits of a display system that uses three GALs to implement the complete decoding and display-driving functions. The inputs to each section are a hex code on pins D_0–D_3, a ripple blanking signal, an intensity control signal, and a lamp test signal.

The hex codes are decoded to form the seven-segment patterns shown in Table 8.1. The input codes, digit represented, and segments driven are as follows:

TABLE 8.1. Function Description

D_3	D_2	D_1	D_0	Digit	Segments
0	0	0	0	0	ABCDEF
0	0	0	1	1	BC
0	0	1	0	2	ABDEG
0	0	1	1	3	ABCDG
0	1	0	0	4	BCFG
0	1	0	1	5	ACDFG
0	1	1	0	6	ACDEFG
0	1	1	1	7	ABC
1	0	0	0	8	ABCDEFG
1	0	0	1	9	ABCDFG
1	0	1	0	A	ABCEFG
1	0	1	1	B	CDEFG
1	1	0	0	C	ADEF
1	1	0	1	D	BCDEG
1	1	1	0	E	ADEFG
1	1	1	1	F	AEFG

Ripple-blanking input RBI is used to suppress leading zeroes in the display. The signal is propagated from the most significant digit to the least significant digit. If the digit input is zero and RBI is low (indicating that the previous digit is also zero), all segments are left blank and this digit position's ripple-blanking output RBO is set low.

Intensity control signal IC controls the duty cycle of the display driver. When IC is high, all segment drivers are turned off. Pulsing this pin with a duty-cycled signal allows the adjustment of the display's apparent brightness.

Lamp test signal LT lets you check to see if all LED segments are energized.

GAL Device Implementation

The GAL16V8 has both the required I/O pins and the drive current capability to perform as the complete display decoder-driver circuit with seven inputs and eight outputs. The logic equations for this circuit are shown in the listing. One GAL device drives each digit; they may be cascaded without limit. With minor changes, the same logical structure could be used with multiplexer logic to allow a single GAL device to decode and drive multiple digits.

8.0 Hexadecimal 7-Segment Display Encoder (Continued)

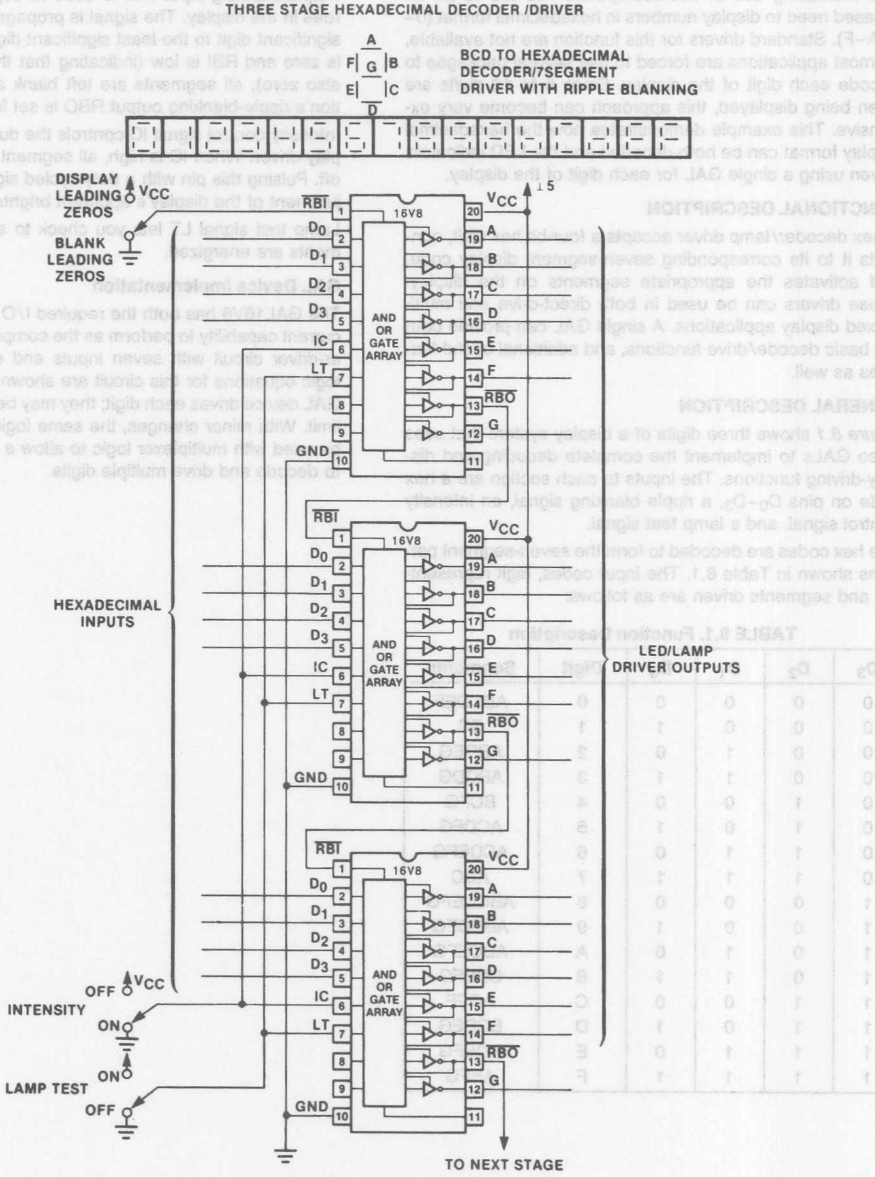
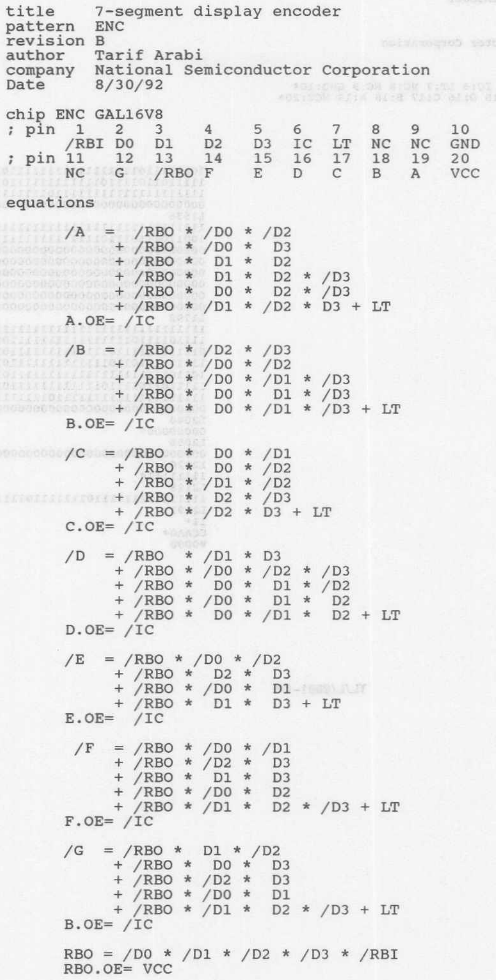


FIGURE 8.1. Hex Display Decoder-Driver Combinational Logic Diagram

8.0 Hexadecimal 7-Segment Display Encoder (Continued)



```
title    7-segment display encoder
pattern  ENC
revision B
author   Tarif Arabi
company  National Semiconductor Corporation
Date     8/30/92

chip ENC GAL16V8
; pin  1   2   3    4    5   6   7   8   9   10
      /RBI D0  D1   D2   D3  IC  LT  NC  NC  GND
; pin 11   12  13   14   15  16  17  18  19  20
     NC    G   /RBO F    E   D   C   B   A   VCC

equations

     /A   = /RBO * /D0 * /D2
          + /RBO * /D0 *  D3
          + /RBO *  D1 *  D2
          + /RBO *  D1 *  D2 * /D3
          + /RBO *  D0 *  D2 * /D3
          + /RBO * /D1 * /D2 * D3 + LT
     A.OE= /IC

     /B   = /RBO * /D2 * /D3
          + /RBO * /D0 * /D2
          + /RBO * /D0 * /D1 * /D3
          + /RBO *  D0 *  D1 * /D3
          + /RBO *  D0 * /D1 * /D3 + LT
     B.OE= /IC

     /C   = /RBO  *  D0 * /D1
          + /RBO  *  D0 * /D2
          + /RBO  * /D1 * /D2
          + /RBO  *  D2 * /D3
          + /RBO  * /D2 * D3 + LT
     C.OE= /IC

     /D   = /RBO  * /D1 * D3
          + /RBO * /D0 * /D2 * /D3
          + /RBO *  D0 *  D1 * /D2
          + /RBO * /D0 *  D1 *  D2
          + /RBO *  D0 * /D1 *  D2 + LT
     D.OE= /IC

     /E   = /RBO * /D0 * /D2
          + /RBO *  D2 *  D3
          + /RBO * /D0 *  D1
          + /RBO *  D1 *  D3 + LT
     E.OE=  /IC

      /F  = /RBO * /D0 * /D1
          + /RBO * /D2 *  D3
          + /RBO *  D1 *  D3
          + /RBO * /D0 *  D2
          + /RBO * /D1 *  D2 * /D3 + LT
     F.OE= /IC

     /G   = /RBO *  D1 * /D2
          + /RBO *  D0 *  D3
          + /RBO * /D2 *  D3
          + /RBO * /D0 *  D1
          + /RBO * /D1 *  D2 * /D3 + LT
     B.OE= /IC

     RBO = /D0 * /D1 * /D2 * /D3 * /RBI
     RBO.OE= VCC
```

TL/L/9991-G4

8.0 Hexadecimal 7-Segment Display Encoder (Continued)

```
        title    7-segment display encoder
        pattern  ENC
        revision B
        author   Tarif Arabi
        company  National Semiconductor Corporation
        Date     8/30/92

        *
NOTE PINS /RBI:1 D0:2 D1:3 D2:4 D3:5 IC:6 LT:7 NC:8 NC:9 GND:10*
NOTE PINS NC:11 G:12 /RBO:13 F:14 E:15 D:16 C:17 B:18 A:19 VCC:20*
QF2194*QP20*F0*
L0000
11111111111111101111111111111111
10111111101111111111111111011111
10111111111101111111111111011111
11110111011111111111111111011111
11110111011110111111111111011111
01111111011110111111111111011111
11111011101101111111111111011111
11111111111111111111110111111111*
L0256
11111111111111101111111111111111
11111111101110111111111111011111
10111111101111111111111111011111
10111011111110111111111111011111
01110111111110111111111111011111
01111011111110111111111111011111
11111111111111111111110111111111
00000000000000000000000000000000*
L0512
11111111111111101111111111111111
01111011111111111111111111011111
01111111101111111111111111011111
11111011101111111111111111011111
11111111011110111111111111011111
11111111101101111111111111011111
11111111111111111111110111111111
00000000000000000000000000000000*
L0768
11111111111111101111111111111111
11111011111110111111111111011111
10111111101110111111111111011111
01110111101111111111111111011111
10110111011111111111111111011111
01111011011111111111111111011111
11111111111111111111110111111111
00000000000000000000000000000000*
L1024
11111111111111101111111111111111
10111111101111111111111111011111
11111111011101111111111111011111
10110111111111111111111111011111
11110111111110111111111111011111
11111111111111111111110111111111
00000000000000000000000000000000
00000000000000000000000000000000*
L1280
11111111111111101111111111111111
10111011111111111111111111011111
11111111101101111111111111011111
11110111111110111111111111011111
```

TL/L/9991-G6

```
10111111101111111111111111011111
11111011011110111111111111011111
11111111111111111111110111111111
00000000000000000000000000000000*
L1536
11111111111111111111111111111111
10011011101110111111111111111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
L1792
11111111111111111111111111111111
11110111101111111111111111011111
01111111111101111111111111011111
11111111101101111111111111011111
10110111111111111111111111011111
11111011011110111111111111011111
11111111111111111111110111111111
00000000000000000000000000000000*
L2048
00000000*
L2056
0000000000000000000000000000000000000000000000000000000000000000*
L2120
11111111*
L2128
1111111111111110111111101111111011111100111111101100000011111110*
L2192
11*
CCAA0*
♥0000
```

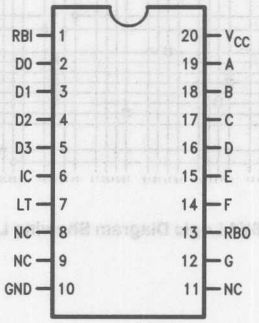
TL/L/9991-61

8.0 Hexadecimal 7-Segment Display Encoder (Continued)

```
Device: 16V8

Pin   Label             Type
---   -----             ----
1     /RBI              neg,com input
2     D0                pos,com input
3     D1                pos,com input
4     D2                pos,com input
5     D3                pos,com input
6     IC                pos,com input
7     LT                pos,com input
8     NC                unused
9     NC                unused
10    GND               ground pin
11    NC                unused
12    G                 neg,trst,com output
13    /RBO              neg,trst,com feedback
14    F                 neg,trst,com output
15    E                 neg,trst,com output
16    D                 neg,trst,com output
17    C                 neg,trst,com output
18    B                 neg,trst,com output
19    A                 neg,trst,com output
20    VCC               power pin

Device Utilization:

No of dedicated inputs used                  :  7/10 (70.0%)
No of dedicated outputs used                 :  2/2  (100.0%)
No of feedbacks used as dedicated outputs    :  5/6  (83.3%)
No of feedbacks used                         :  1/6  (16.7%)

                ------------------------------------------
                Pin   Label                 Terms  Usage
                ------------------------------------------
                19    A.oe                  1/1    (100.0%)
                19    A                     6/7    (85.7%)
                18    B.oe                  1/1    (100.0%)
                18    B                     6/7    (85.7%)
                17    C.oe                  1/1    (100.0%)
                17    C                     5/7    (71.4%)
                16    D.oe                  1/1    (100.0%)
                16    D                     5/7    (71.4%)
                15    E.oe                  1/1    (100.0%)
                15    E                     4/7    (57.1%)
                14    F.oe                  1/1    (100.0%)
                14    F                     5/7    (71.4%)
                13    RBO.oe                1/1    (100.0%)
                13    RBO                   1/7    (14.3%)
                12    G.oe                  1/1    (100.0%)
                12    G                     6/7    (85.7%)
                ------------------------------------------
                Total Terms                51/64   (79.7%)
                ------------------------------------------
```

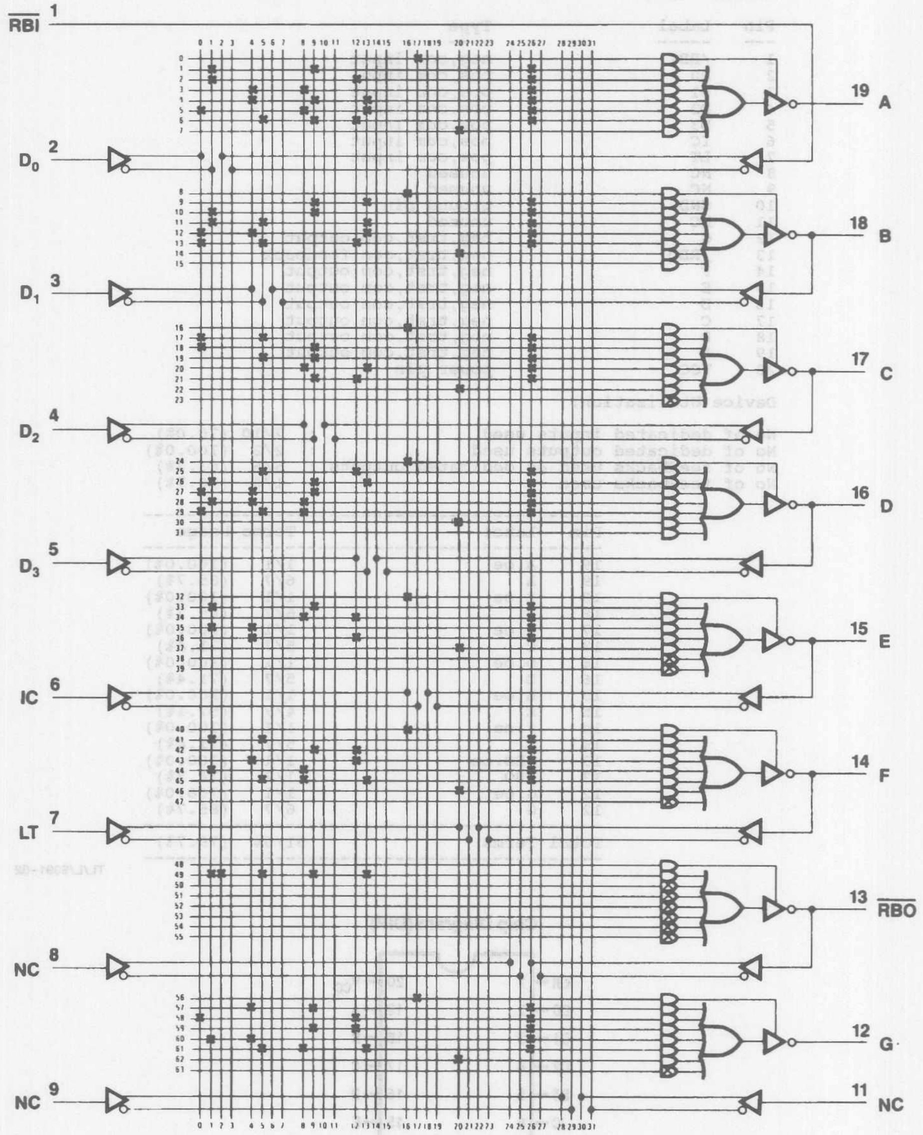
TL/L/9991-62

Chip Diagram (DIP)



TL/L/9991-60

8.0 Hexadecimal 7-Segment Display Encoder (Continued)



TL/L/9991-72

FIGURE 8.2. GAL16V8 Logic Diagram Showing Lamp Driver Pattern

the GAL16V8, a dual-port, dynamic-RAM controller capable of controlling four banks of DRAMs is implemented. The design, whose block diagram is shown in *Figure 9.1* and state diagrams in *Figures 9.2* and *9.3*, requires two GAL16V8 devices. The CUPL input listings for each device are shown in *Figures 9.4* and *9.6*, with respective simulation files shown in *Figures 9.5* and *9.7*.

The first device, the Controller, is primarily responsible for maintaining the state of the entire circuit. As shown by its state diagram in *Figure 9.2*, the Controller normally resides in the IDLE state. It can cycle to any of the states: RFGT (Refresh Grant), RQGTA (Request Grant A), or RQGTB (Request Grant B), depending on the inputs: REFRQ (Refresh Request), MRQA (Memory Request A), or MRQB (Memory Request B).

REFRQ has top priority, since the refresh cycle is of vital importance for DRAM memory retention. MRQA is arbitrarily chosen to have priority over MRQB to avoid bus contention with contiguous requests. Every REQUEST, whether a refresh request or a memory request, must receive an ACK (acknowledge) signal before the Controller will continue to cycle. Once an ACK is received, the Controller will either return to the IDLE state or perform a refresh (if REFRQ is present), and then return to the IDLE state. Cycling between RQGTA and RQGTB is also possible.

The CUPL input file for the Controller, shown in *Figure 9.4*, distinguishes output declarations from intermediate variable definitions, which greatly reduce the complexity of declarations. BK_3–BK_0 are intermediate definitions decoded from address lines A_{17} and A_{16} to determine which bank used to simplify the final output declarations.

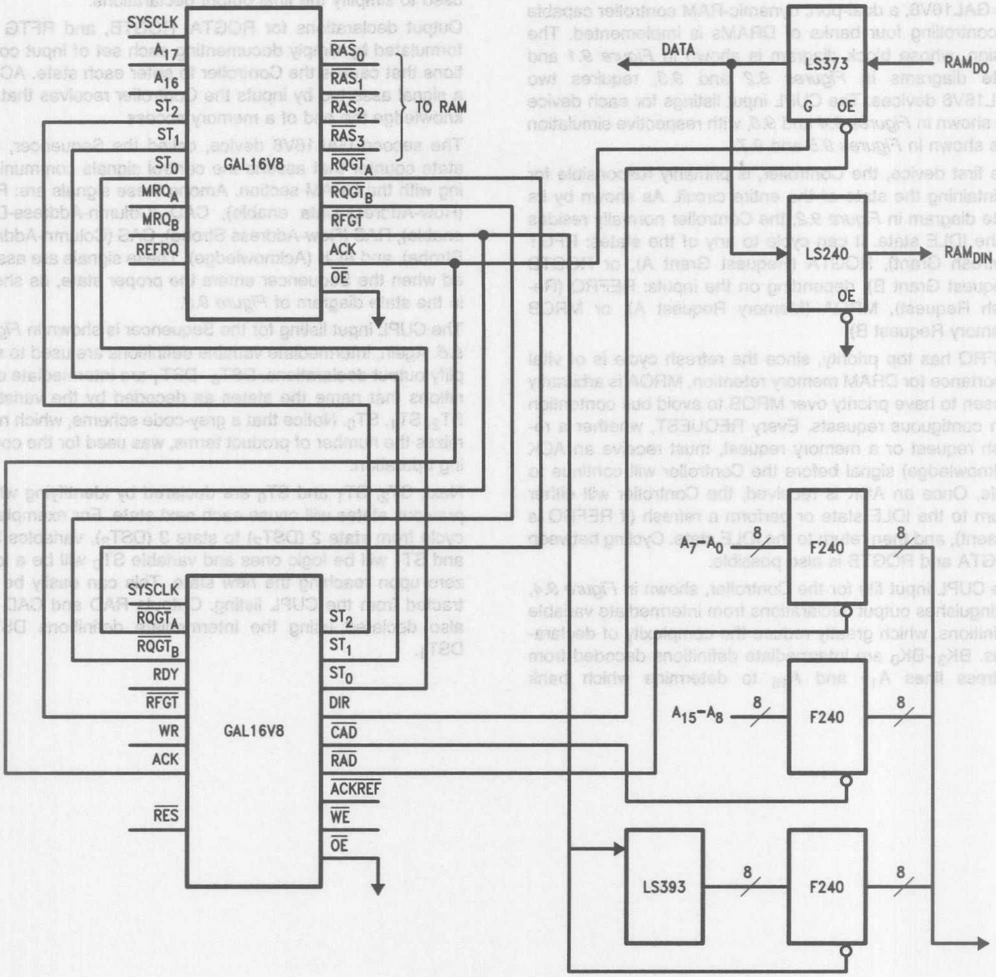
Output declarations for RQGTA, RQGTB, and RFTG are formulated by simply documenting each set of input conditions that causes the Controller to enter each state. ACK is a signal asserted by inputs the Controller receives that acknowledge the end of a memory access.

The second GAL16V8 device, called the Sequencer, is a state counter that asserts the control signals communicating with the DRAM section. Among these signals are: RAD (Row-Address-Data enable), CAD (Column-Address-Data enable), RAS (Row-Address Strobe), CAS (Column-Address Strobe), and ACK (Acknowledge). These signals are asserted when the Sequencer enters the proper state, as shown in the state diagram of *Figure 9.3*.

The CUPL input listing for the Sequencer is shown in *Figure 9.6*. Again, intermediate variable definitions are used to simplify output declarations. DST_8–DST_1 are intermediate definitions that name the states as decoded by the variables ST_2, ST_1, ST_0. Notice that a grey-code scheme, which minimizes the number of product terms, was used for the counting operation.

Next, ST_2, ST_1 and ST_0 are declared by identifying which previous states will cause each next state. For example, to cycle from state 2 (DST_2) to state 3 (DST_3), variables ST_2 and ST_1 will be logic ones and variable ST_0 will be a logic zero upon reaching the new state. This can easily be extracted from the CUPL listing. Outputs RAD and CAD are also declared using the intermediate definitions DST_8–DST_1.

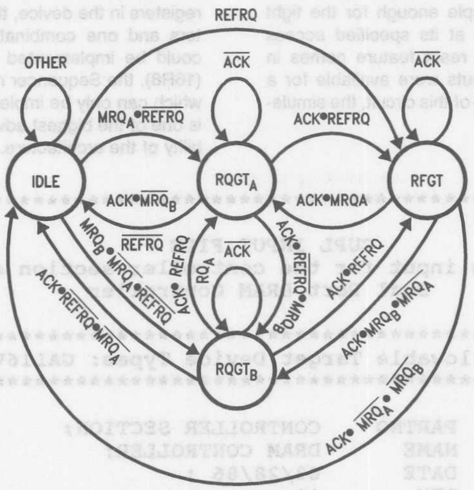
9.0 Dual-Port RAM Controller (Continued)



TL/L/9991–74

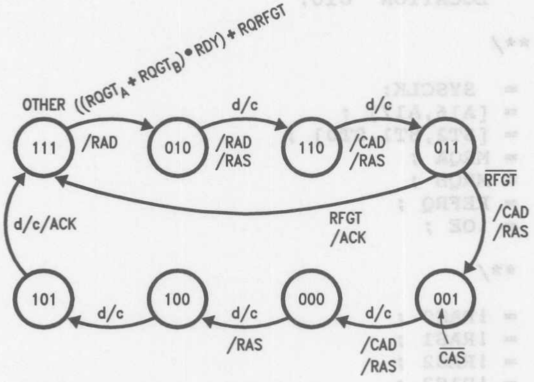
FIGURE 9.1. Block Diagram

9.0 Dual-Port RAM Controller (Continued)



TL/L/9991-75

FIGURE 9.2. State Diagram for Controller Section



TL/L/9991-76

FIGURE 9.3. State Diagram for Sequencer Section

9.0 Dual-Port RAM Controller (Continued)

The two GAL16V8-25 devices can be clocked at cycle times as fast as 35 ns (28.5 MHz), ample enough for the tight timings required to run a DRAM at its specified access times. The GAL16V8's power-up reset feature comes in handy in this circuit, since no inputs were available for a reset term. To test the functionality of this circuit, the simulation facilities of CUPL were used.

It should be noted that although the Controller uses all eight registers in the device, the Sequencer requires seven registers and one combinational output. While the Controller could be implemented in a traditional PAL configuration (16R8), the Sequencer requires a nonstandard architecture which can only be implemented in a GAL16V8 device. This is one of the biggest advantages of GAL devices—the flexibility of the architecture.

```
/******************************************************************/
/*                                                                */
/*                      CUPL INPUT FILE                           */
/*        Design input for the controller section of the          */
/*              Dual Port DRAM Controller                         */
/*                                                                */
/******************************************************************/
/*          Allowable Target Device Types: GAL16V8                */
/******************************************************************/

                PARTNO    CONTROLLER SECTION;
                NAME      DRAM CONTROLLER;
                DATE      03/28/86 ;
                REV       01 ;
                DESIGNER  Joe Engineer;
                COMPANY   National Semiconductor;
                ASSEMBLY  ONE;
                LOCATION  U10;

/**  Inputs  **/

PIN 1         =  SYSCLK;
PIN [2,3]     = [A16,A17] ;
PIN [4..6]    = [ST2,ST1,ST0] ;
PIN 7         = MRQA ;
PIN 8         = MRQB ;
PIN 9         = REFRQ ;
PIN 11        = !OE ;

/**  Outputs  **/

PIN 19        = !RAS0 ;
PIN 18        = !RAS1 ;
PIN 17        = !RAS2 ;
PIN 16        = !RAS3 ;
PIN 15        = !RQGTA ;
PIN 14        = !RQGTB ;
PIN 13        = !RFGT ;
PIN 12        = ACK ;
```

TL/L/9991-77

FIGURE 9.4. Design Input File for Controller Section

```
/** Declarations and Intermediate Variable Definitions **/

BK0 = (!A17 & !A16) # RFGT ;
BK1 = (!A17 & A16) # RFGT ;
BK2 = (A17 & !A16) # RFGT ;
BK3 = (A17 & A16) # RFGT ;

RASEN = !ST2 & ST1 & !ST0 # ST2 & ST1 & !ST0 # !ST2 & ST1 & ST0 #
        !ST2 & !ST1 & ST0 # !ST2 & !ST1 & !ST0 ;

RAS0.D = BK0 & RASEN ;
RAS1.D = BK1 & RASEN ;
RAS2.D = BK2 & RASEN ;
RAS3.D = BK3 & RASEN ;

RQGTAS = RQGTA & !RQGTB & !RFGT ;

RQGTBS = !RQGTA & RQGTB & !RFGT ;

RFGTS = !RQGTA & !RQGTB & RFGT ;

IDLE = !RQGTA & !RQGTB & !RFGT ;

RQGTA.D = (IDLE & MRQA & !REFRQ # RQGTAS & !ACK # RQGTBS & ACK &
        MRQA & !REFRQ  # RFGTS & ACK & MRQA) & !(ACK & !MRQA &
        !MRQB & !REFRQ) ;

RQGTB.D = (IDLE & !MRQA & !REFRQ & MRQB # RQGTBS & !ACK # RQGTAS &
        ACK & MRQB & !REFRQ # RFGTS & ACK & !MRQA & MRQB) & !(ACK &
        !MRQA & !MRQB & !REFRQ) ;

RFGT.D = (IDLE & REFRQ # RFGTS & !ACK # RQGTAS & ACK & REFRQ #
        RQGTBS & ACK & REFRQ) & !(ACK & !MRQA & !MRQB & !REFRQ) ;

ACK.D = ST2 & !ST1 & ST0 # !ST2 & ST1 & ST0 & RFGT ;
```

TL/L/9991-78

FIGURE 9.4. Design Input File for Controller Section (Continued)

9.0 Dual-Port RAM Controller (Continued)

```
/****************************************************************/
/*                                                              */
/*                    CUPL SIMULATION FILE                      */
/*      Simulation input for the controller section of the      */
/*                 Dual Port DRAM Controller                    */
/*                                                              */
/****************************************************************/
/*            Allowable Target Device Types: GAL16V8            */
/****************************************************************/

                PARTNO    CONTROLLER SECTION;
                NAME      DRAM CONTROLLER;
                DATE      03/28/86 ;
                REV       01 ;
                DESIGNER  Joe Engineer;
                COMPANY   National Semiconductor;
                ASSEMBLY  ONE;
                LOCATION  U10;

ORDER:
SYSCLK,%2,A17,A16,%2,ST2,ST1,ST0,%2,MRQA,MRQB,REFRQ,%2,!OE,%4,
!RAS0,!RAS1,!RAS2,!RAS3,%2,!RQGTA,!RQGTB,!RFGT,%2,ACK;

VECTORS:
$msg" S                          !!    ";
$msg" Y             R      !!!!  RR!   ";
$msg" S           MME      RRRR  QQR   ";
$msg" C AA  SSS   RRF  !   AAAA  GGF  A ";
$msg" L 11  TTT   QQR  O   SSSS  TTG  C ";
$msg" K 76  210   ABQ  E   0123  ABT  K ";
$msg" --------------------------------- ";

      0 00  101   000  0   XXXX  XXX  X
      C 00  101   000  0   HHHH  XXX  H
      C 00  111   000  0   HHHH  HHH  L
      C 00  111   000  0   HHHH  HHH  L
      C 00  111   100  0   HHHH  LHH  L
      C 00  010   100  0   LHHH  LHH  L
      C 00  110   100  0   LHHH  LHH  L
      C 00  011   100  0   LHHH  LHH  L
      C 00  001   100  0   LHHH  LHH  L
      C 00  000   100  0   LHHH  LHH  L
      C 00  100   100  0   HHHH  LHH  L
      C 00  101   110  0   HHHH  LHH  H
      C 00  111   110  0   HHHH  HLH  L
      C 11  111   010  0   HHHH  HLH  L
      C 11  111   010  0   HHHH  HLH  L
      C 11  010   010  0   HHHL  HLH  L
      C 11  110   010  0   HHHL  HLH  L
      C 11  011   010  0   HHHL  HLH  L
      C 11  001   010  0   HHHL  HLH  L
      C 11  000   000  0   HHHL  HLH  L
      C 11  100   101  0   HHHH  HLH  L
      C 11  101   101  0   HHHH  HLH  H
      C 00  111   101  0   HHHH  HHL  L
      C 00  111   101  0   HHHH  HHL  L
      C 11  010   000  0   LLLL  HHL  L
      C 11  110   000  0   LLLL  HHL  L
      C 11  011   000  0   LLLL  HHL  H
      C 11  001   000  0   LLLL  HHH  L
      C 11  000   000  0   HHHL  HHH  L
      C 11  100   101  0   HHHH  HHL  L
      C 11  101   101  0   HHHH  HHL  H
      C 00  111   101  0   HHHH  LHH  L
      C 00  111   101  0   HHHH  LHH  L
```

TL/L/9991-80

FIGURE 9.5. Simulation File for Controller Section

9.0 Dual-Port RAM Controller (Continued)

```
/*******************************************************************/
/*                                                                 */
/*                     CUPL INPUT FILE                             */
/*        Design input for the sequencer section for the           */
/*                   Dual Port DRAM Controller                     */
/*                                                                 */
/*******************************************************************/
/*           Allowable Target Device Types: GAL16V8                */
/*******************************************************************/

                PARTNO    SEQUENCER SECTION;
                NAME      DRAM CONTROLLER;
                DATE      03/28/86 ;
                REV       01 ;
                DESIGNER  Joe Engineer;
                COMPANY   National Semiconductor;
                ASSEMBLY  TWO;
                LOCATION  U11;

/**  Inputs  **/

PIN 1          =  SYSCLK;
PIN [2,3]      = [!RQGTA,!RQGTB] ;
PIN 4          = RDY ;
PIN 5          = !RFGT ;
PIN 6          = !WR ;
PIN 7          = ACK ;
PIN 8          = !RES ;
PIN 11         = !OE ;
/**  Outputs  **/

PIN 19         = ST2 ;
PIN 18         = ST1 ;
PIN 17         = ST0 ;
PIN 16         = DIR ;
PIN 15         = !CAD ;
PIN 14         = !RAD ;
PIN 13         = !ACKREF ;
PIN 12         = !WE ;
```

TL/L/9991-81

FIGURE 9.6. Input File for Sequencer Section

```
DST1 = ST2 & ST1 & ST0 ;
DST2 = !ST2 & ST1 & !ST0 ;
DST3 = ST2 & ST1 & !ST0 ;
DST4 = !ST2 & ST1 & ST0 ;
DST5 = !ST2 & !ST1 & ST0 ;
DST6 = !ST2 & !ST1 & !ST0 ;
DST7 = ST2 & !ST1 & !ST0 ;
DST8 = ST2 & !ST1 & ST0 ;

STCYC = ((RQGTA # RQGTB) & RDY # RFGT) ;

ST2.D = (DST2 # DST6 # DST8 # DST7 # DST4 & RFGT) # RES #
         DST1 & !STCYC ;

ST1.D = DST2 # DST3 # DST8 # DST4 & RFGT # DST1 # RES ;

ST0.D = (DST3 # DST4 # DST7 # DST8) # RES # DST1 & !STCYC ;

DIR.D = WR & !DST1 ;

CAD.D = DST3 & !RFGT # DST4 & !RFGT # DST5 ;

RAD.D = (RQGTA # RQGTB) & RDY & (DST1 # DST2) ;

ACKREF = RFGT & ACK ;

WE.D = WR & (DST5 # DST6) ;
```

TL/L/9991-82

FIGURE 9.6. Input File for Sequencer Section (Continued)

```
/******************************************************************/
/*                                                                */
/*                   CUPL SIMULATION FILE                         */
/*     Simulation File for the sequencer section of the           */
/*                 Dual Port DRAM Controller                      */
/*                                                                */
/******************************************************************/
/*         Allowable Target Device Types: GAL16V8                 */
/******************************************************************/

                PARTNO      SEQUENCER SECTION;
                NAME        DRAM CONTROLLER;
                DATE        03/28/86 ;
                REV         01 ;
                DESIGNER    Joe Engineer;
                COMPANY     National Semiconductor;
                ASSEMBLY    TWO;
                LOCATION    U11;

ORDER:
SYSCLK,%2,!RES,%2,!RQGTA,!RQGTB,!RFGT,%2,RDY,!WR,ACK,%2,!OE,%4,
ST2,ST1,ST0,%2,DIR,%2,!CAD,!RAD,%2,!WE ;

VECTORS:

$num" S     !!                          ";
$num" Y     RR!                         ";
$num" S  !  QQR                  !!     ";
$num" C  R  GGF  R!A  !    SSS  D  CR  ! ";
$num" L  E  TTG  DWC  O    TTT  I  AA  W ";
$num" K  S  ABT  YRK  E    210  R  DD  E ";
$num" ----------------------------------";
      0  0  111  010  0    XXX  X  XX  X
      0  0  111  010  0    XXX  X  XX  X
      C  0  111  010  0    HHH  L  XH  H
      C  1  011  010  0    HHH  L  HH  H
      C  1  011  110  0    LHL  L  HL  H
      C  1  011  110  0    HHL  L  HL  H
      C  1  011  010  0    LHH  L  LH  H
      C  1  011  010  0    LLH  L  LH  H
      C  1  011  010  0    LLL  L  LH  H
      C  1  011  010  0    HLL  L  HH  H
      C  1  011  010  0    HLH  L  HH  H
      C  1  011  010  0    HHH  L  HH  H
      C  1  011  010  0    HHH  L  HH  H
      C  1  011  010  0    HHH  L  HH  H
```

TL/L/9991-83

FIGURE 9.7. Simulation File for Sequencer Section

10.0 CPU Board Random Control Logic

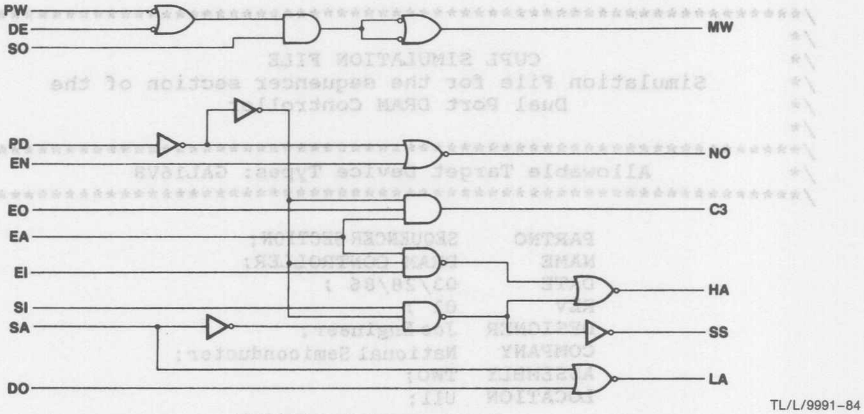


FIGURE 10. CPU Control Logic

10.0 CPU Board Random Control Logic (Continued)

OPAL™ INPUT FILE

```
title     CPU CONTROL LOGIC
pattern   CPU
revision  B
author    Tarif Engineer
company   NSC
Date      1/8/92

chip CPU GAL16V8

PD EN E0 EA S1 SA E1 D0 DE GND
S0 NC N0 C3 HA SS LA MW PW VCC

equations

  MW = /S0 + PW * DE
  LA = /SA * /D0
  SS = S1 * PD * /SA
  HA = S1 * PD * /SA *EA * E1
  C3 = PD * E0 * EA
  N0 = PD * /EN
```

TL/L/9991-65

OPAL™ JEDEC FILE

```
☻
GAL16V8
EQN2JED - Boolean Equations to JEDEC file assembler (Version V029)
Copyright (c) National Semiconductor Corporation 1990,1991
Assembled from "C:\OPAL102\CPU.EQN". Date: 1-8-92
        title     CPU CONTROL LOGIC
        pattern   CPU
        revision  B
        author    Tarif Engineer
        company   NSC
        Date      1/8/92

        *
NOTE PINS PD:1 EN:2 E0:3 EA:4 S1:5 SA:6 E1:7 D0:8 DE:9 GND:10*
NOTE PINS S0:11 NC:12 N0:13 C3:14 HA:15 SS:16 LA:17 MW:18 PW:19*
NOTE PINS VCC:20*
QF2194*QP20*F0*
L0256
11111111111111111111111111111110
11111011111111111111111111110111*
L0512
11111111111111111011111110111111*
L0768
11011111111101111011111111111111*
L1024
11011111011101111011011111111111*
L1280
11010111011111111111111111111111*
L1536
10011111111111111111111111111111*
L2048
01111110*
L2056
0000000000000000000000000000000000000000000000000000000000000000*
L2120
10000000*
L2128
0000000011000000100000001000000010000000100000001000000000000000*
L2192
10*
C1B40*
♥0000
```

TL/L/9991-85

High-Speed System Design Using Programmable Generic Array Logic (GAL®) Devices

National Semiconductor
Application Note 707
Tarif Arabi
Senior Applications Engineer

TABLE OF CONTENTS

1.0 High Speed Consideration

2.0 Ground Bounce

- **Defining the Problem**
- **Analyzing the Problem**
- **Solution**

3.0 Interfacing with TTL, CMOS and FACT Family

4.0 Decoupling Problem

5.0 Capacitive Load vs Propagation Delay

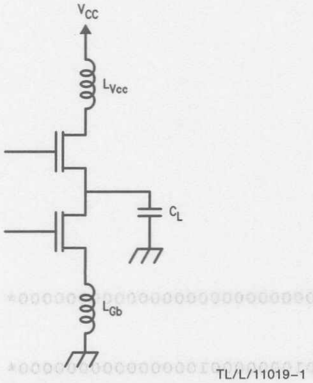
6.0 Terminating Transmission Line

7.0 High Speed Design Rules

1.0 HIGH SPEED CONSIDERATIONS

Today's system designer is faced with the problem of keeping ahead when addressing system performance and reliability. With the development of high speed, fast slew rate, TTL compatible GAL devices, high speed system design is becoming more complicated and needs more attention and analysis.

This application note will describe the high speed design issues and possible solutions that can be used to eliminate them. The above items will be discussed and analyzed to give the user an understanding of the problems involved and help in finding the best solution.



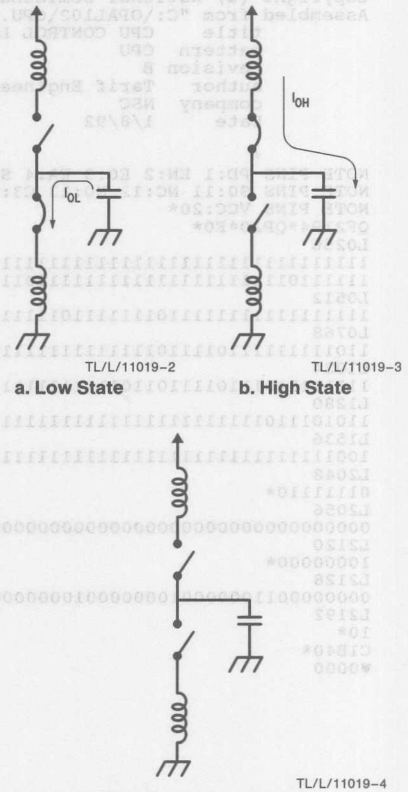
TL/L/11019–1

FIGURE 1. GAL Output Structure

2.0 GROUND BOUNCE

Defining the Problem

To understand the ground bounce problem we will analyze the output structure of the GAL device. The output of the GAL consists of two MOS transistors, acting as fast switches to give the desired output transition function going from high to low or low to high, or from TRI-STATE® (high impedance) to either high or low. *Figure 1* shows the output structure of the GAL device, and *Figure 2* shows the simplified models of the GAL device as it implements the various output functions. *Figure 2a* shows the output at logic LOW (GND), where lower transistor is ON and the upper transistor is OFF. *Figure 2b* shows the output at logic HIGH (V_{CC}), where the lower transistor is OFF and the upper transistor is ON. *Figure 2c* shows the high impedance state (TRI-STATE), where both transistors are OFF.



TL/L/11019–2

a. Low State

TL/L/11019–3

b. High State

TL/L/11019–4

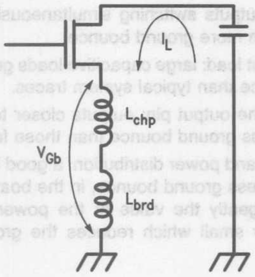
c. TRI-STATE (High-Impedance)

FIGURE 2

The ground bounce problem starts when the output is switching from HIGH to LOW. In this case the lower transistor is turned from the "OFF" state to the "ON" state, causing high current to pass through the transistor and the ground lead.

Analyzing the Ground Bounce Problem

If we take the simple model for the current path, as shown in *Figure 3* and put in the equations to realize the effect of this transition, then the current can be calculated as follows:



TL/L/11019–5

FIGURE 3

$$I = C_L \bullet dV/dt$$

where

$$C_L = C_{OUT} + C_{IN1} + C_{IN2} + \ldots + C_{INM}$$ (as shown in *Figure 4*)

Let us assume $C_L = 50$ pF and dV/dt for fast edge GAL devices = 1.5V/1 ns

Then $I = 50 \bullet 10^{-12} \bullet (1.5/10^{-9}) = 75$ mA

The current will enter the low state loop and will cause a difference in voltage between the power supply ground and the device ground, V_{gb}. We can calculate this voltage:

$$|+V_{gb}| = |-V_{gb}| = L_{gb} \bullet |dI/dt|$$

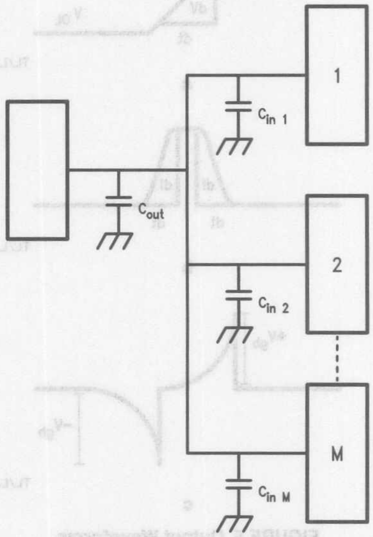
where; $L_{gb} = L_{chp} + L_{brd}$

L_{chp} is the inductance of the ground lead inside the device

L_{brd} is the inductance of the ground lead on the board

let us assume $L_{gb} = 20$ nH (for both the board and the device) and assume that the current will change from 0 mA to 75 mA or from 75 mA to 0 mA in 2 ns so dI/dt = 75 mA/2 ns.

Then $V_{gb} = 20 \bullet 10^{-9} \bullet 75 \bullet 10^{-3}/2 \bullet 10^{-9}$
$= 750$ mV



TL/L/11019–6

FIGURE 4. Load Capacitance

The three waveforms shown in *Figure 5* depict how the ground bounce is generated. The first waveform (*5a*) shows the voltage (V) across the load as it is switched from a logic HIGH to a logic LOW. The second waveform (5b) shows the current that is generated as the load capacitor discharges (where $I = C_L$ dV/dt), and the third waveform (*5c*) shows the voltage that is induced across the inductance in the ground lead.

Note: These models are included in order to generate an understanding of where ground bounce originates. To get accurate results many other factors must be taken into account which would result in some highly complex equations, and also the actual waveforms are dependant on where the measurements are taken.

When more outputs switch simultaneously the current going through the ground lead increases, causing a greater voltage drop across the ground inductance, and generating higher ground bounce.

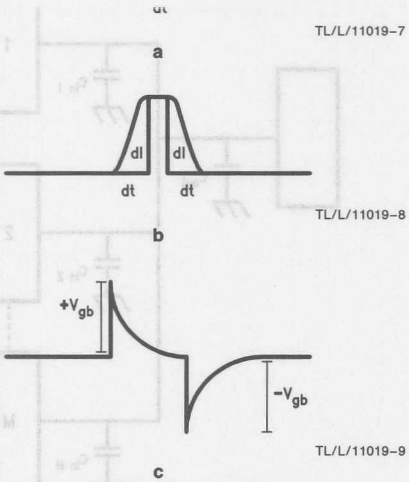


FIGURE 5. Output Waveforms

The ground bounce in a GAL device might cause the following problem:

1. Glitch on the quiet LOW output:

   When the output is quiet LOW (lower transistor is ON) the output voltage is referenced to the device ground, and any shift in the device ground will look like a shift in the output voltage as shown in *Figure 6.* If this output is driving another logic device this glitch might cause a false signal because the overshoot of the ground bounce (the positive pulse) might be considered as a HIGH signal at the input of the driven device.

2. False input:

   The input buffers of the GAL device are also referenced to the same output ground, so a shift in the ground voltage may change the input thresholds so as to cause a false transition. If this input is a clock, it might cause a new state in a GAL device operating in registers mode.

The overshoot and the undershoot that generated by the ground bounce are considered as a source of a noise in the system.

From the previous discussion we can see that there are many factors that affect the amplitude of the ground bounce:

1. Number of outputs switching simultaneously: more outputs results in more ground bounce.
2. Type of output load: large capacitive loads generate more ground bounce than typical system traces.
3. Location of the output pin: outputs closer to the ground pin exhibit less ground bounce than those further away.
4. Board layout and power distribution: a good board design will result in less ground bounce, in the board that is designed intelligently the value of the power lead inductance is very small which reduces the ground bounce problem.

Solution

Now that the problem has been defined, we will discuss the methods for solving ground bounce induced phenomena. This can be further sub-divided into two parts: I—*The Manufacturing Solution* (over which the device designer has control), and II—*The Board Design Solution* (over which the system designer has control).

I—Manufacturing Solution

1. Slow the output buffer switching speed to reduce the edge rate (dI/dt) and so reduce ground bounce. Make the transition from HIGH to LOW slower by using a number of MOS transistors. Put a switching delay between each one so that the first one will open, then the second one, and so on, as shown in *Figure 7a* and *7b*. This will help only if the board is designed correctly. In a poorly designed board, where L_{brd} is greater than L_{chp}, this solution will not make a difference.

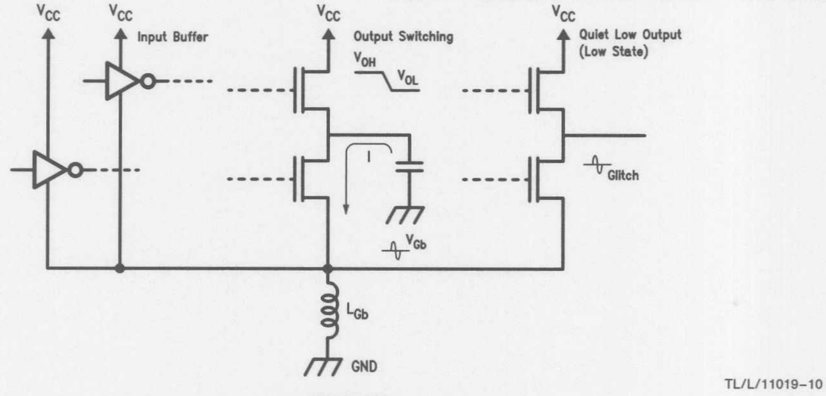


FIGURE 6

2. Isolate the output ground from the ground of all other elements in the device by having a package with a split ground lead. *Figures 8* and *9* show the difference between the regular ground lead and the split ground lead. This method will cancel the effect of the (L_{chp} • dI/dt) on the input, because the output current will only go in the dedicated output ground lead. The only effect on the input will be the product of L_{brd} • dI/dt. Careful board design will be the key issue in the ground bounce problem. Note that the ground split solution will not help in reducing the noise on the quiet LOW outputs, since they are sharing the same ground lead, as shown before in *Figure 6*. Also, note that the ground split solution is not effective in a PLCC package, since the ground lead is centered.

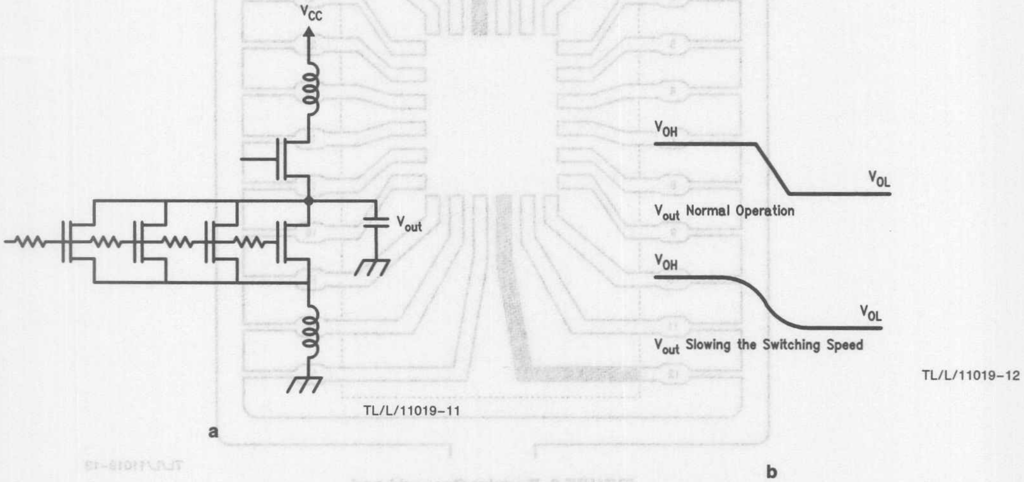
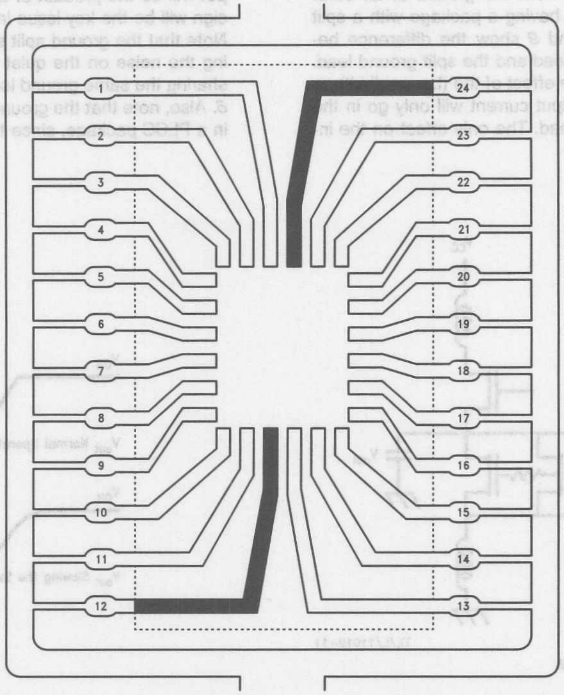
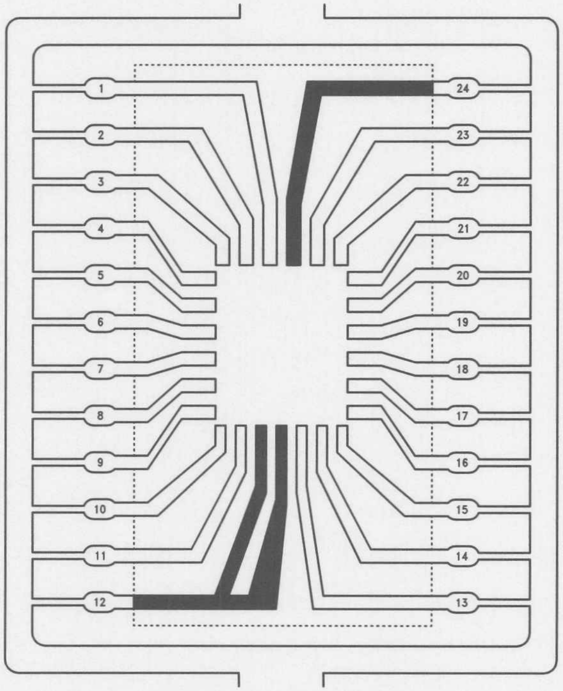


FIGURE 7



TL/L/11019–13

FIGURE 8. Regular Ground Lead



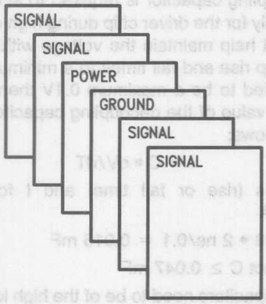
TL/L/11019–14

FIGURE 9. Split Ground Lead

II—Board Design Solution

As we see from the previous discussion, an important factor of the ground bounce problem is a good system and board design. There are rules the system designer should follow to obtain high speed system and to minimize the problems of designing with next-generation logic. The following rules should be considered when designing with high speed GAL devices:

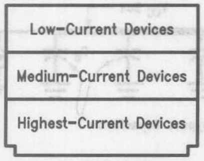
1. Ground planes and power planes are required, as shown in *Figure 10*. This implies a multi-layer board is also required. Use multi-layer boards with V_{CC} and ground planes, with the device power pins soldered directly to the planes to insure the lowest ground and V_{CC} line inductances possible.



TL/L/11019–15

FIGURE 10. Typical Multilayer Printed Circuit Board

2. Devices driving the highest current should be as close as possible to the power entry point as shown in *Figure 11*.



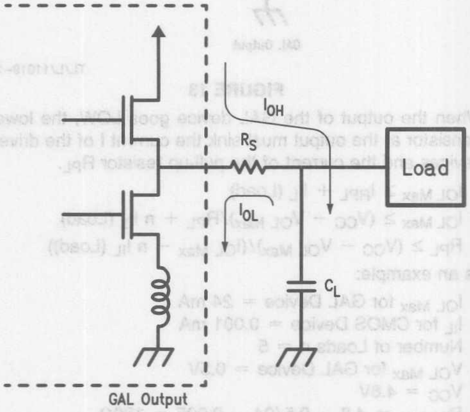
TL/L/11019–16

FIGURE 11. Group HIGH Current Devices Near Power Entry Point

3. Use decoupling capacitor for every device, usually 0.1 mF should be adequate. These capacitors should be located as close to the ground pin as possible.
4. Do not use sockets or wirewrap boards to guarantee good connection in the board and reduce any possible impedance.
5. Do not exceed the manufacturer's limit in capacitive loading of the outputs.
6. Minimize the number of outputs switching simultaneously from high to low if possible.
7. Use a serial resistor R_S connected between the output and the load for the outputs that drive more current, to minimize the current during transitions, as shown in *Figure 11*. This resistor will affect the propagation delay of the GAL device. To calculate the extra delay, let us consider $R_S = 10\Omega$. The time constant of the loop is:

$$T = R_S \bullet C_L = 10 \bullet 50 \text{ pF} = 0.5 \text{ ns}$$

This value is small compared to the propagation delay of the device. At the same time the series resistor will limit the switching current.



TL/L/11019–17

FIGURE 12

3.0 INTERFACING WITH OTHER LOGIC FAMILIES:

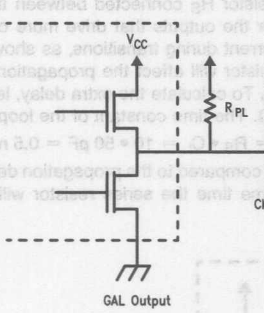
An interfacing problem starts when the output logic level or the current requirements of the driver is different than the input logic levels or the current of the driven device. The most important parameters in this case are V_{IH}, V_{IL} I_{IL}, I_{IH} for the driven device and V_{OH}, V_{OL}, I_{OH}, I_{OL} for the driver device. The design engineer will face the problem when the different logic families are not compatible. Modification of the circuit should be done to achieve compatibility.

I—Interfacing GAL Devices to the TTL Logic Family

An EECMOS GAL device can drive a TTL device wihout additional circuitry. This is because the GAL device specifications are fully compatible with those of TTL devices and vice versa.

II—Interfacing GAL Devices to the CMOS Logic Family

The voltage levels of the two families are not compatible. When interfacing the GAL as a driver to a CMOS device, a problem arises because $V_{OH\ Min}$ for GAL = 2.4V, while $V_{IH\ Min}$ for CMOS = 4V. To solve this problem, a pull-up resistor R_{PL} should be used as shown in *Figure 13*.



TL/L/11019–18

FIGURE 13

When the output of the GAL device goes LOW, the lower transistor at the output must sink the current I of the driven devices and the current of the pull-up resistor R_{PL}.

$I_{OL\ Max} \geq I_{RPL} + I_{IL}$ (Load)

$I_{OL\ Max} \geq (V_{CC} - V_{OL\ Max})/R_{PL} + n\ I_{IL}$ (Load)

$R_{PL} \geq (V_{CC} - V_{OL\ Max})/(I_{OL\ Max} - n\ I_{IL}$ (Load))

as an example:

$I_{OL\ Max}$ for GAL Device = 24 mA
I_{IL} for CMOS Device = 0.001 mA
Number of Loads n = 5
$V_{OL\ Max}$ for GAL Device = 0.5V
V_{CC} = 4.8V
$R_{PL\ Min} = 4.8 - 0.5/24 - 0.005 = 180\Omega$

This defines the lower limit for the pull-up resistor.

Two factors affect the upper limit of the pull-up resistor R_{PL}:

1. Load high level input current.
2. Load input capacitance.

When the GAL output goes HIGH the I_{IH} current of the load device must not cause a voltage drop across R_{PL} such that V_{IH} is violated so:

$V_{CC} - n \bullet I_{IH} \bullet R_{PL} \geq V_{IH\ Min}$

$R_{PL} \leq V_{CC} - V_{IH\ Min} / n\ I_{IH}$

as an example

V_{CC} = 4.8V
V_{IH} = 4V
I_{IH} = 0.001 mA
n = 5

$R_{PL\ Max} = 160\ k\Omega$

The other factor is that the input of the CMOS device will rise exponentially with a time constant equal to ($R_{PL} \bullet C_L$):

$V_{IH\ Min} = V_{CC}\ [1 - e^{-(t/R_{PL} \bullet C_L)}]$

$R_{PL} = t/k \bullet C_L$

where $k = -1n\ [(V_{CC} - V_{IH})/V_{CC}] = 1.8$

Maximum Input Rise or Fall Time for CMOS Device = 400 ns

so $R_{PL\ Max} = 400\ \text{ns}/(1.8 \bullet 50\ \text{pF}) = 4.44\ k\Omega$

(Assume the load capacitor is 50 pF)

Usually the rise time is the main factor for the highest value of R_{PL}.

III—Interfacing GAL Devices to the FACT™ Logic Family

The voltage levels of these two families are also incompatible. When interfacing the GAL as a driver to a FACT device, the problem arises because $V_{OH\ Min}$ for GAL = 2.4V while $V_{IH\ Min}$ for FACT = 3.5V so the same solution for interfacing GAL devices to CMOS devices can use for interfacing GAL devices to the FACT family.

4.0 DECOUPLING REQUIREMENTS

GAL devices, like other high-drive, high-performance, logic families have special requirements for decoupling and board layout. When the output of the GAL switches the device draws a substantial supply current, which will produce a current spike on the V_{CC} and GND leads of the device.

A local decoupling capacitor is required to act as a low impedance supply for the driver chip during high current conditions. This will help maintain the voltage within acceptable limits and keep rise and fall times to a minimum. If the V_{CC} drop is specified to be a maximum 0.1V then as shown in *Figure 14* the value of the decoupling capacitor can be calculated as follows:

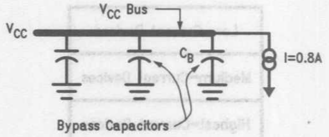
$$I = C \bullet dV/dT$$

If dt = 2 ns (rise or fall time) and I for the power supply = 0.8A

then $C = 0.8 \bullet 2\ \text{ns}/0.1 = 0.016\ \text{mF}$

so select $C \geq 0.047$ mF

Decoupling capacitors need to be of the high k ceramic type with low equivalent series resistance (ESR). Capacitors using 5ZU dielectric have suitable properties and make a good choice for decoupling capacitors. They offer minimum cost and effective performance.



TL/L/11019–19

FIGURE 14

Powering GAL Devices and Connecting Unused Inputs

When GAL devices are powered-up the following steps should be followed to avoid damaging the devices:

1. Connect ground first
2. Connect V_{CC}
3. Connect the input signal

When powering down, the reverse order of the above should be followed.

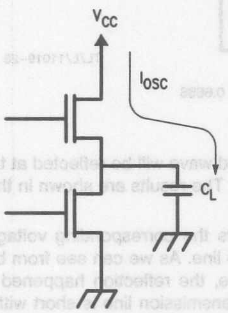
To avoid any noise and reduce power consumption the unused inputs should be tied to either V_{CC} or ground.

5.0 CAPACITIVE LOADING EFFECTS ON PROPAGATION DELAY

One of the factors that affects the propagation delay (t_{PD}) is the capacitive load C_L. The additional delay caused by C_L can be calculated if the short circuit current on the output is known.

Since the maximum current on the output is the short circuit current so as shown in *Figure 15*, the current can be calculated as:

$I_{OS} = C_L \, dV/dt$



TL/L/11019–20

FIGURE 15

The propagation delay is measured to the 1.5V point of the output waveform:

$$dt = C_L(1.5V)/I_{OS}$$

This equation gives the general form of the additional propagation delay caused by changing the capacitive load from the specified load in data sheet. To calculate the propagation delay for a specific load capacitance (C_L), the following equation may be used

where $t_{PDT} = t_{PD} + 1.5\ (C_L - 50\ pF)/I_{OS}$

t_{PDT} = Total propagation delay

t_{PD} = Specified propagation delay for 50 pF load

C_L = Actual load capacitance I_{OS} = Short circuit current

as an example:

t_{PD} (50 pF) = 10 ns

C_L = 150 pF

I_{OS} = 80 mA

t_{PDT} (150 pF) = 10 + (1.5)(150 pF − 50 pF)/80)

= 10 ns + 1.9 ns = 11.9 ns

The above equations and the example are used here as a design aid, not as a guarantee.

6.0 TERMINATION FOR HIGH-SPEED GAL DEVICES

One common characteristic shared by high speed logic is that the output edge rates are very fast. As all other logic families, careful consideration must be made to determine if termination is necessary, and what is the best termination to be used. Any connection between two devices in a high speed design should be considered as a transmission line. That is because of the high ratio of rise time to the line propagation delay time. In order to obtain a pure signal, the line length must be very short compared to the signal rise time. Otherwise proper termination must be used.

Reflections on the transmission line are caused by discontinuity in line impedance, which is considered a major source of noise in high speed digital systems. Discontinuity can be caused by an input device, another circuit, a connector, or another transmission line.

The reflection coefficient (r) is the ratio of the voltage in the reflected wave to that in the incident wave

$$r = Vr/Vi$$

The signal that flows through the line at the discontinuity is V. The summation of the two waves (the incident and the reflected waves)

$$V = Vi + Vr$$

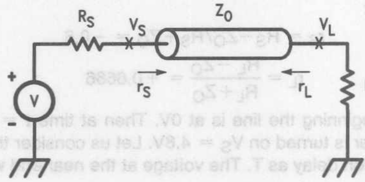
The reflection coefficient is given by the relation:

$$r = (Z_L - Z_O)/(Z_L + Z_O)$$

where Z_L is the Load impedance and Z_O is the line impedance.

Reflection Diagram

Consider a basic transmission line as shown in *Figure 16*. The circuit has a source signal with impedance = R_S, a line impedance of Z_O and the load impedance is R_L.

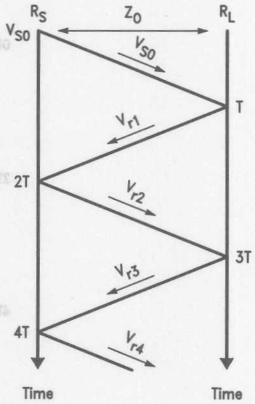


TL/L/11019–21

$$r_S = \frac{R_S - Z_O}{R_S + Z_O} \qquad r_L = \frac{R_L - Z_O}{R_L + Z_O}$$

FIGURE 16

A reflection diagram is commonly used to demonstrate the reflection in the line where, as shown in *Figure 17*, the vertical axes are the time scales and the graph shows the signal going back and forth along the transmission line. We can see that after one line propagation delay (T), the input voltage (Vs0) will reach the end of the transmission line and reflect back with the new voltage $V_{R1} = r_1\ V_{S0}$. The output of the line (the load side) will see the value as ($V_{L1} = V_{S0} + V_{R1}$). This new voltage will act as an input to the line and after (2T) this voltage will reach the other end of the line and reflect again with a value of $V_{R2} = r_2\ V_{OL}$ which will change the voltage at the source side and reflect again going back and forth.



TL/L/11019–22

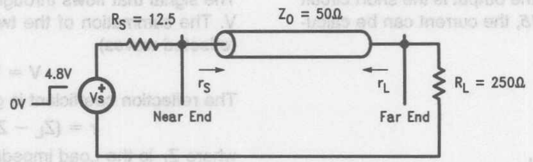
$V_{S2} = V_{S0}\ (1 + r_L + r_S\ r_L)$

$V_{S4} = V_{S0}\ (1 + r_L + r_S\ r_L + r_S\ r_L^2 + r_S^2\ r_L^2)$

$V_{L1} = V_{S0} + r_L\ V_{S0}$

$V_{L3} = V_{S0}\ (1 + r_L + r_S\ r_L + r_S\ r_L^2)$

FIGURE 17



TL/L/11019-23

$$r_S = \frac{12.5 - 50}{12.5 + 50} = -0.6 \qquad r_L = \frac{250 - 50}{250 + 50} = 0.6666$$

FIGURE 18

Let us consider a line, shown in *Figure 18*, as an example to calculate the reflection. Assume the value R_S = 12.5 Ω, Z_O = 50Ω and R_L = 250Ω. The reflection coefficients at the ends are:

$$r_S = R_S - Z_O / R_S + Z_O = -0.6$$

$$r_L = \frac{R_L - Z_O}{R_L + Z_O} = +0.6666$$

At the beginning the line is at 0V. Then at time t = 0 the transmitter is turned on V_S = 4.8V. Let us consider the line propagation delay as T. The voltage at the near end will be:

$$V_{S0} = \frac{V_S\, Z_O}{R_S + Z_O} = 4.8(50/62.5) = 4V$$

A voltage wave V_{S0} = 4V travels toward the far end which will be reflected at the far end at time t = T with reflection coefficient = r_L giving the reflected wave V_{R1} as:

$$V_{R1} = V_{S0} \bullet r_L = 4 \bullet 0.6666 = 2.67$$

The load voltage (voltage at the far end) will be:

$$V_{L1} = 4 + 2.67 = 6.67V$$

The backward wave V_{R1} returns to the transmitter and reaches it at t = 2T where it reflects and generates a new forward wave

$$V_{R2} = V_{R1} \bullet r_S = -1.6V$$

The new voltage at the transmitter is equal to the old value V_{S0} plus the backward wave V_{R1} plus the outgoing forward wave V_{R2}

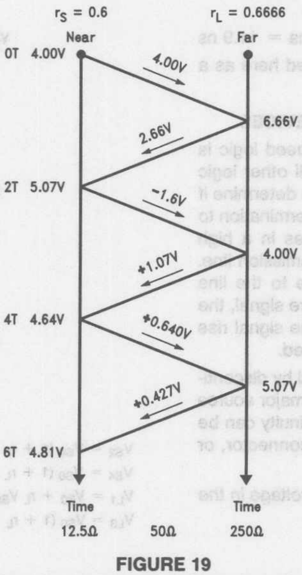
$$V_{S2} = V_{S0} + V_{R1} + V_{R2} = 4 + 2.67 - 1.6 = 5.07V$$

This new forward wave will be reflected at t = 3T at the far end, and so on. The results are shown in the reflection diagram *Figure 19*.

Figure 20 shows the corresponding voltages observed at each end of the line. As we can see from both ends of the transmission line, the reflection happened each period of time T. If the transmission line is short with respect to the edge rate of the incident signal, then reflections will have no effect on the quality of the incident signal. A good rule in this cause is that terminating the transmission line is necessary if the line delay exceeds one quarter the rise or fall time of the incident signal. For typical transmission lines in today's systems this translates to approximately six to eight inches (15 cm to 20 cm).

The main two effects of the unterminated transmission line are the overshoot and the undershoot of the transmitted signal as shown in *Figure 16*. The degree of the undershoot or the overshoot depends on the ratio of the edge speed to the propagation delay of the line.

It is important to differentiate between the undershoot and overshoot generated by the driving device's output and that generated by reflected energy. Many times undershoot and overshoot is the result of reflected energy and not caused by the driving devices directly (ground bounce problem). Poor termination might cause undershoot and overshoot to exceed 2V.



TL/L/11019-24

FIGURE 19

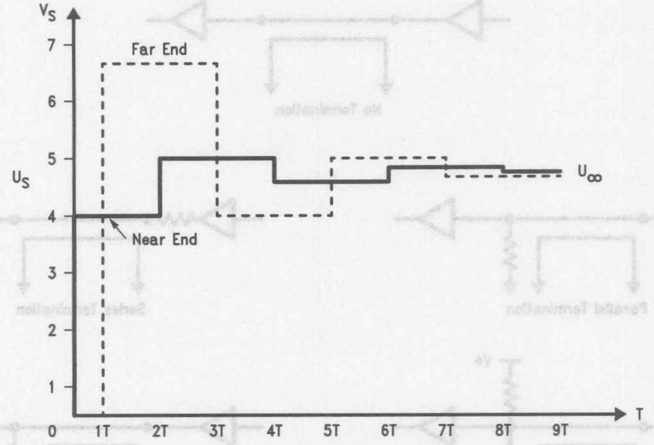
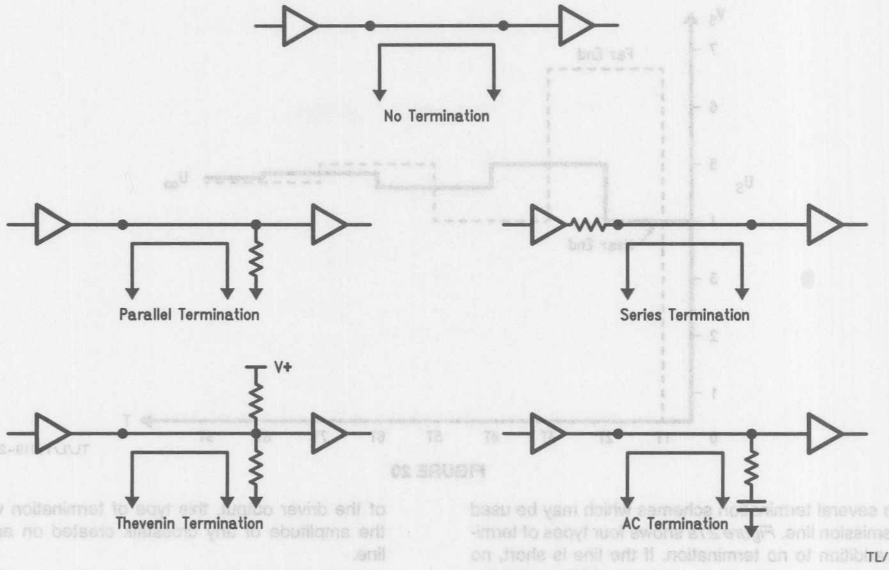


FIGURE 20

There are several termination schemes which may be used on a transmission line. *Figure 21a* shows four types of termination in addition to no termination. If the line is short, no termination may be necessary, and the ESD protection structures on the inputs of the GAL devices will clamp the undershoot.

1. Parallel Termination: Parallel termination is not generally recommended for CMOS devices due to the power consumption. The power consumption of parallel termination is a function of the resistor value and the duty cycle of the signal.
2. Series Termination: Series termination is most useful in high speed applications where most of the line loads are at the far end of the line. This type of termination is recommended for CMOS devices, as it offers the lowest-power consumption termination. Series termination divides the line voltage in half. The reflection created at the end of the line doubles the voltage back up to the original value. Since the voltage swing along the line is one-half of the driver output, this type of termination will reduce the amplitude of any crosstalk created on an adjacent line.
3. Thevenin Termination: Thevenin termination is also not generally recommended for CMOS devices due to power consumption. Thevenin termination is much like parallel termination except that it has a balanced effect on the output voltage. Lines terminated with a Thevenin termination should not be floated or TRI-STATEd otherwise the receivers along that line may oscillate.
4. AC Termination: AC termination works well for applications where the delay caused by series terminations is unacceptable. In this type of termination the capacitor blocks any DC current path and help to reduce power consumption. This termination technique is recommended for transmission lines driving distributed receivers.

Figure 21b shows the suggested termination values for the four types of above termination techniques.



TERMINATION VALUES

Parallel:	Resistor = Z_O
Thevenin:	Resistor = 2 x Z_O
Series:	Resistor = $Z_O - Z_{OUT}$
AC:	Resistor = Z_O
	Capacitor = $C \geq \frac{3t_d}{Z_O}$

Where:	Z_O	is the line impedance
	Z_{OUT}	is the output driver impedance
	t_d	is the line propagation delay

FIGURE 21

7.0 HIGH-SPEED PROGRAMMABLE LOGIC DESIGN RULES AND RECOMMENDATIONS

The following rules should be followed when designing with high speed digital logic:

1. In a high speed circuit, a multilayer PCB is required.
2. Separate ground planes and power planes are required.
3. Highest current devices should be placed as close as possible to the power entry point.
4. Use decoupling capacitors for every device in a high-speed circuit. The capacitor should be located as close to the ground pin as possible.
5. Minimize the number of outputs switching simultaneously from high to low.
6. Avoid discontinuity in ground plane and transmission lines and avoid sharp bends in the transmission line.
7. Do not exceed the manufacturer's limitation in the output capacitive load.
8. Use serial termination for high-speed outputs.
9. Terminate all high-speed signal lines.
10. Avoid using wirewrap boards and sockets.
11. Power pins should be soldered directly to the power planes.

PAL® to GAL® Conversion

National Semiconductor
Application Note 799

TABLE OF CONTENTS

1.0 INTRODUCTION
2.0 CONVERSION PROCESS
3.0 TECHNICAL DIFFERENCES BETWEEN PAL AND GAL DEVICES
4.0 CONVERTING TO GAL22V10
5.0 CONVERTING AN XOR PAL TO A GAL22V10
6.0 PAL TO GAL CROSS REFERENCE
7.0 APPENDIX: OPAL™/OPALjr FLOWS

1.0 INTRODUCTION

Every day more designers are recognizing the advantages of EECMOS GAL devices over TTL PAL devices (PAL and GAL will henceforth imply TTL and EECMOS respectively). Some of the advantages are:

- GAL16V8, GAL20V8 and GAL22V10 replace most PAL devices
  —Reduced inventory, testing, etc.
- EECMOS technology
  —Reprogrammable, instant erase
  —100% factory tested means higher reliability, yield
  —Low power consumption
- GAL Quiet Series™ have guaranteed low noise specifications

In most applications GAL devices can be used in place of PAL devices, however, a simple conversion process is necessary. Most people are aware of the advantages of GAL devices but are not fully aware of the subtle differences. This application note will try to explain the conversion process and highlight some of the important differences between PAL and GAL devices.

2.0 CONVERSION PROCESS

A conversion process is needed to convert a PAL JEDEC file to a GAL JEDEC file. The difference in JEDEC files represents differences in device architecture. The conversion process can be accomplished using two simple methods.

- Cross programming method
- Software conversion method

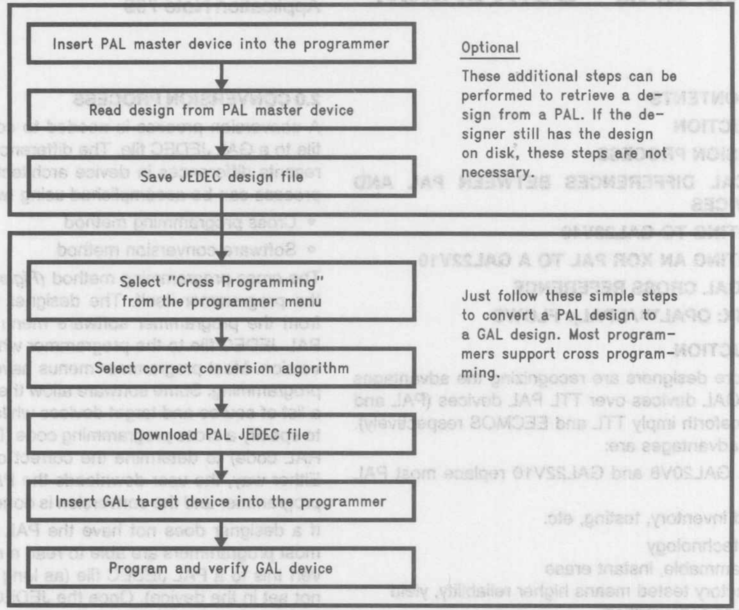
The cross programming method *(Figure 1)* is performed by the programmer itself. The designer selects this function from the programmer software menu and downloads the PAL JEDEC file to the programmer which handles the conversion. Most programmer menus have an option for cross programming. Some software allow the user to choose from a list of source and target devices while others ask the user to specify a cross programming code (Data I/O® calls this a RAL code) to determine the correct conversion algorithm. Either way, the user downloads the PAL JEDEC file to the programmer and the conversion is done by the programmer.

If a designer does not have the PAL design files on disk, most programmers are able to read a master PAL and convert this to a PAL JEDEC file (as long as the security bit is not set in the device). Once the JEDEC file information has been saved on disk, the conversion process proceeds as outlined above.

The software conversion method is also very simple. Most PLD design software including National Semiconductor's OPAL and OPALjr PLD Design Software allow the software conversion of a PAL JEDEC file to a GAL JEDEC file.

Figures 2 and *3* show the software conversion method using OPAL or OPALjr. The PAL2GAL tool is selected from the "MODULES" menu *(Figure 4)* and the source and target JEDEC files are specified. Upon hitting return, a GAL JEDEC file is created which can be downloaded to the programmer containing the GAL device. The new GAL JEDEC file uses the filename extension .GJD to differentiate it from the old PAL JEDEC file with extension .JED.

METHOD 1: CROSS PROGRAMMING


Insert PAL master device into the programmer
Read design from PAL master device
Save JEDEC design file
Select "Cross Programming" from the programmer menu
Select correct conversion algorithm
Download PAL JEDEC file
Insert GAL target device into the programmer
Program and verify GAL device
Optional

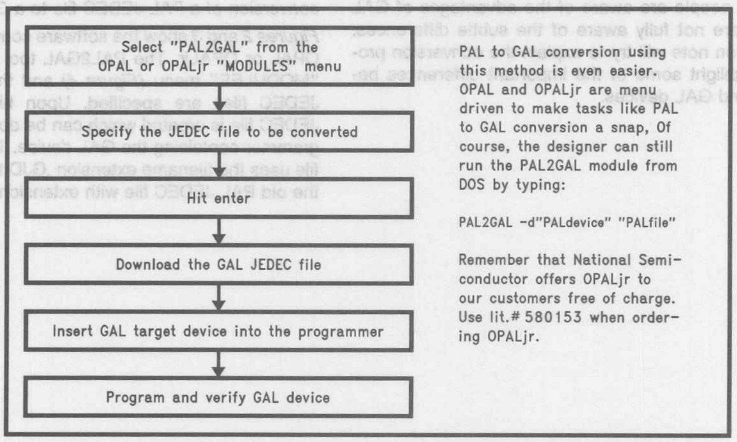
These additional steps can be performed to retrieve a design from a PAL. If the designer still has the design on disk, these steps are not necessary.

Just follow these simple steps to convert a PAL design to a GAL design. Most programmers support cross programming.

TL/L/11293–1

FIGURE 1. The cross programming method

METHOD 2: SOFTWARE CONVERSION


Select "PAL2GAL" from the OPAL or OPALjr "MODULES" menu
Specify the JEDEC file to be converted
Hit enter
Download the GAL JEDEC file
Insert GAL target device into the programmer
Program and verify GAL device

PAL to GAL conversion using this method is even easier. OPAL and OPALjr are menu driven to make tasks like PAL to GAL conversion a snap, Of course, the designer can still run the PAL2GAL module from DOS by typing:

```
PAL2GAL -d"PALdevice" "PALfile"
```

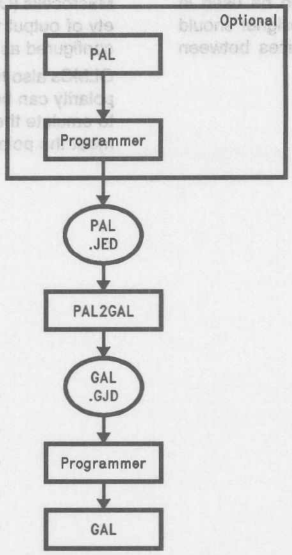
Remember that National Semiconductor offers OPALjr to our customers free of charge. Use lit.# 580153 when ordering OPALjr.

TL/L/11293–2

FIGURE 2. The software conversion method using OPAL or OPALjr

METHOD 2: SOFTWARE CONVERSION (Continued)

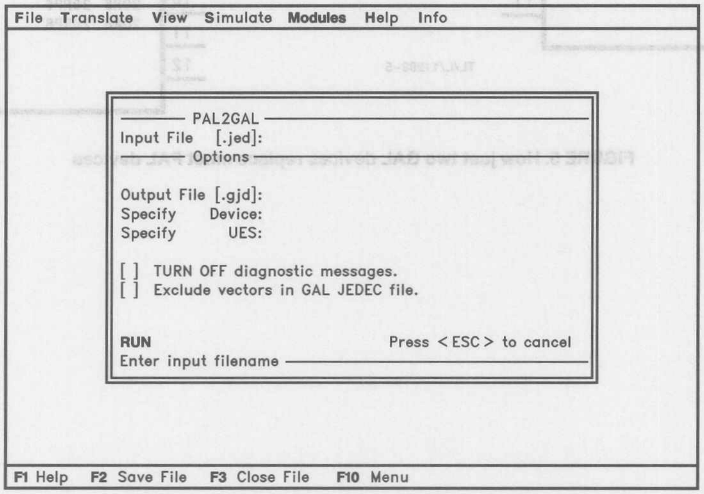
OPAL/OPALjr PAL to GAL Conversion Flow



TL/L/11293–3

FIGURE 3. PAL to GAL software flow using OPAL or OPALjr

OPAL/OPALjr PAL2GAL Menu


File Translate View Simulate Modules Help Info
F1 Help F2 Save File F3 Close File F10 Menu

```
PAL2GAL
Input File   [.jed]:
       Options

Output File [.gjd]:
Specify     Device:
Specify        UES:

[ ]  TURN OFF diagnostic messages.
[ ]  Exclude vectors in GAL JEDEC file.


RUN                            Press <ESC> to cancel
Enter input filename
```

TL/L/11293–4

FIGURE 4. PAL2GAL module menu from OPAL or OPALjr

3.0 TECHNICAL DIFFERENCES BETWEEN PAL AND GAL DEVICES

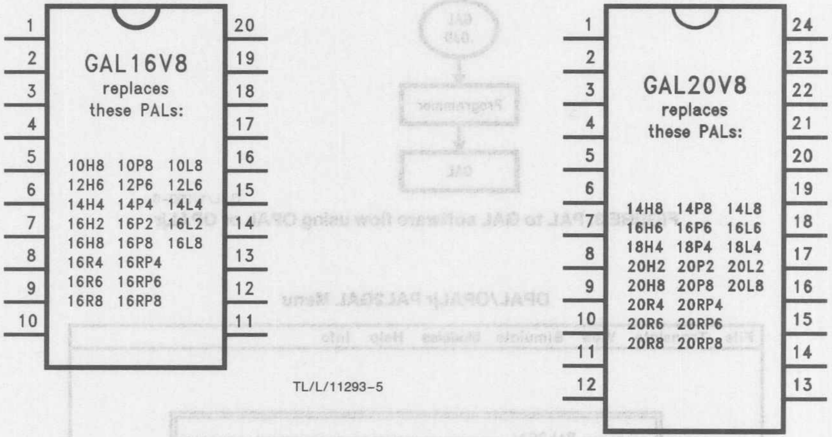
As PLDs have evolved, IC manufacturers have made subtle changes to newer PAL and GAL devices to make these devices more useful. While GAL devices can be used in place of PAL devices in most designs, the designer should be aware of some of the important differences between these two devices.

- Output Logic Macrocells (OLMCs)
- Floating inputs
- Power-up state
- Edge rates
- Reprogrammability
- Power Consumption
- User Electronic Signature (UES)
- Checksum

3.1 Output Logic Macrocells (OLMCs)

PAL devices have many density and output variations. GAL devices, have standardized densities and configurable output structures. These output structures, called Output Logic Macrocells (OLMCs), can be programmed to emulate a variety of output types. *Figures 6* and *7* show how OLMCs are configured as combinatorial and registered outputs.

OLMCs also have programmable output polarity. The output polarity can be programmed either active high or active low to emulate the output polarity of a specific PAL device or to meet the polarity requirements of the system.



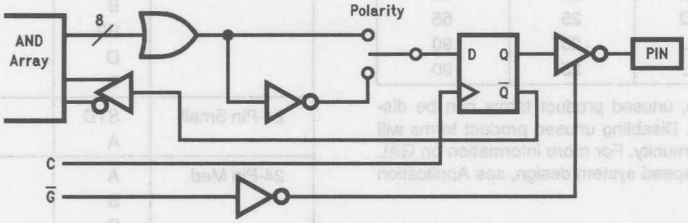
TL/L/11293-5

TL/L/11293-6

FIGURE 5. How just two GAL devices replace most PAL devices

TL/L/11293-7

FIGURE 6. Example OLMC in combinatorial configuration



TL/L/11293-8

FIGURE 7. Example OLMC in registered configuration

3.2 Floating Inputs

GAL and PAL devices have TTL compatible I/O, however, GAL devices use CMOS input and output structures. Therefore, unused GAL inputs must be pulled up or down in order to function properly. In applications where a GAL device will replace a PAL device whose input(s) were left floating:

- Pull the unused inputs up or down with a 1 kΩ to 5 kΩ resistor, or
- Use a GAL22V10 device which has built-in pull-up resistors. GAL22V10 devices are pin-for-pin compatible with GAL20V8 devices.

3.3 Power-Up

Some PAL devices have different power-up reset specifications from GAL devices. Since power-up reset is undefined for most PAL devices, converting to a GAL device is generally not a problem. NSC 24-pin medium B-PAL devices, however, reset to a low state. This may be a problem if power-up state is part of the design.

Additionally, test vectors are sometimes included to test the power-up reset state of the device. These vectors are usually at the start of the test vector file and must be changed to reflect the power-up reset specifications of the new device. Otherwise, the device may seem to fail mysteriously on the programmer as a result of incorrect power-up state test conditions.

3.4 Edge Rates

GAL devices tend to have faster edge rates than TTL PAL devices, which makes them more prone to noise. This can present a problem in some high speed systems. In noise-critical designs, the designer should consider GAL Quiet Series devices. GAL Quiet Series (GAL QS) devices have guaranteed low noise specifications assuring the designer that noise will be within guaranteed limits (four specifications guarantee simultaneous switching noise level and dynamic threshold performance). At the time of this printing, ten GAL QS devices are available from National Semiconductor.

GAL QS Device	t_{PD} (ns)	I_{CC} (mA)
GAL16V8QS-15Q	15	55
GAL20V8QS-15Q	15	55
GAL16V8QS-15L	15	90
GAL20V8QS-15L	15	90
GAL16V8QS-20L	20	90
GAL20V8QS-20L	20	90
GAL16V8QS-25Q	25	55
GAL20V8QS-25Q	25	55
GAL16V8QS-25L	25	90
GAL20V8QS-25L	25	90

To help reduce noise, unused product terms can be disabled in GAL devices. Disabling unused product terms will help improve noise immunity. For more information on GAL QS devices and high-speed system design, see Application Note 707.

3.5 Reprogrammability

One of the greatest benefits of EECMOS GAL devices is reprogrammability. While a minimum of 100 erase/write cycles is guaranteed, many thousands of erase/write cycles may be realized. In addition, the erase cycle is almost instantaneous as compared with UV PAL devices. These facts are well known to system designers, but they also have an impact on system reliability. Since GAL devices can be programmed and tested at the factory, National guarantees 100% field programmability and functionality.

Power-Up Reset Reference Chart

NSC Device		Polarity	Time
20-Pin Small	STD	x	x
	A	x	x
	A2	x	x
20-Pin Med	STD	x	x
	A	x	x
	A2	x	x
	B	x	x
	B2	x	x
	D	1	1 µs
	7	1	1 µs
24-Pin Small	STD	x	x
	A	x	x
24-Pin Med	A	x	x
	B	0	1 µs
	D	1	1 µs
24-Pin XOR	STD	x	x
	A	1	1 µs
24-Pin Pol.	B	1	1 µs
PAL16RA8		1	1 µs
PAL20RA10		1	1 µs
GAL16V8		1	45 µs
GAL20V8		1	45 µs
GAL22V10		1	45 µs
GAL20RA10		1	45 µs
GAL6001		1	45 µs

Note: x = Power-Up reset undefined.

FIGURE 8. Power-up reset conditions

3.6 Power Consumption

Converting from a PAL to a GAL device will reduce power consumption by 50% to 75%.

	PAL			GAL	
	Med 24-Pin	Med 20-Pin	½ Pwr	Low Pwr	Qtr Pwr
I_{CC} (mA)	210	180	90	90	45

3.7 User Electronic Signature (UES)

GAL devices contain a User Electronic Signature (UES) which contains 64-bits of user identification data. The designer may specify any relevant data up to 64-bits such as date, revision, and pattern codes. The UES may be specified in most PLD design software.

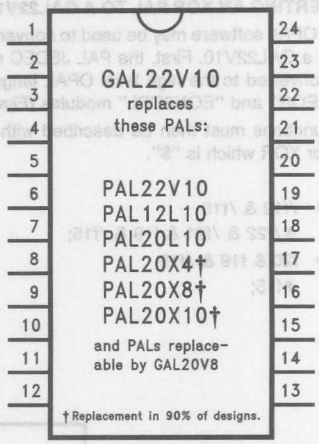
3.8 Checksum

The checksum of identically configured PAL and GAL devices will be different due to the extra programmable cells in GAL devices. This is important to remember since it could potentially cause confusion on the production floor. In addition, data in the UES will affect the checksum. This means that two identical GAL devices may be functionally identical but may have different checksums due to different UES data. This, too, can cause confusion on the production floor. It is recommended that the user set the UES to zeros if it is not used. Most software will program the UES to zeros by default.

4.0 CONVERTING TO GAL22V10

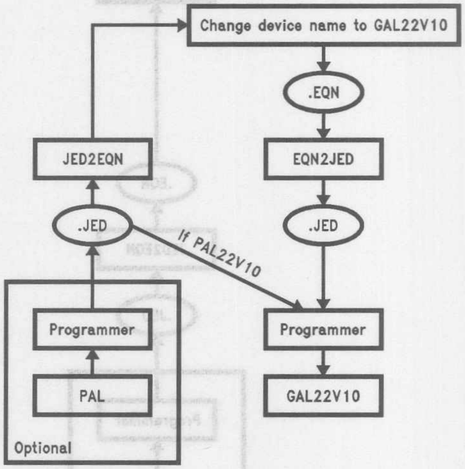
A GAL22V10 will replace most PAL devices including PAL devices replaceable by a GAL20V8 (see *Figure 9*). In order to convert a PAL device to a GAL22V10, OPAL or OPALjr may be used. Start with the PAL JEDEC file and run the module called "JED2EQN" to convert the JEDEC file back to Boolean equations. Next, edit the .EQN file and change the device name from PALxxxxx to GAL22V10, then recompile to a JEDEC file using the "EQN2JED" module. Now the new GAL JEDEC file can be downloaded to the programmer and the GAL22V10 *(Figure 10)*.

Since a PAL22V10 JEDEC file is identical to that of a GAL 22V10 except for the absence of UES, this recompilation is not needed. Programmer software contain two options for programming a GAL22V10. They are labeled "GAL22V10" and "GAL22V10 UES". Choose "GAL22V10" and program the GAL22V10 using the PAL22V10 JEDEC file. *OPAL 2.0 now supports automatic PAL to GAL22V10 conversion.*



TL/L/11293-9

FIGURE 9. GAL22V10 replaces large PAL devices



TL/L/11293-10

FIGURE 10. Converting from PAL to GAL22V10 using OPAL or OPALjr

5.0 CONVERTING AN XOR PAL TO A GAL22V10

National's OPAL software may be used to convert XOR PAL devices to a GAL22V10. First, the PAL JEDEC or EQN file must be converted to the high level OPAL language using the "JED2EQN" and "EQN2OPL" modules *(Figure 11).*

All XOR functions must then be described with the OPAL operator for XOR which is "$".

Example

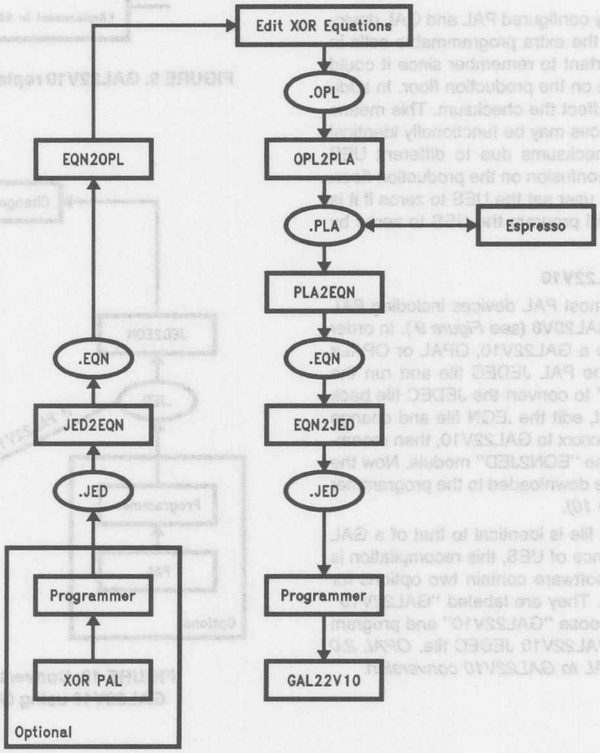
```
f20.X1  :=  /f19 & /f15
            #/f22 & /f21 & f19 & /f15;
f20.X2  :=  f20 & f19 & /f15
            #f15;
```

must be converted to:

```
f20  :=      (/f19 & /f15
              #/f22 & /f21 & f19 & /f15)
           $(f20 & f19 & /f15
              #f15);
```

Note the use of parentheses.

After all XOR functions are converted to the new format, the design should be recompiled to a JEDEC file. OPAL will expand the XOR functions to standard sum-of-products equation form during recompilation. For this reason, the Espresso minimization option should be chosen so that OPAL can attempt to shrink the design to fit in the GAL22V10.



TL/L/11293–11

FIGURE 11. Converting an XOR PAL to a GAL22V10 using OPAL

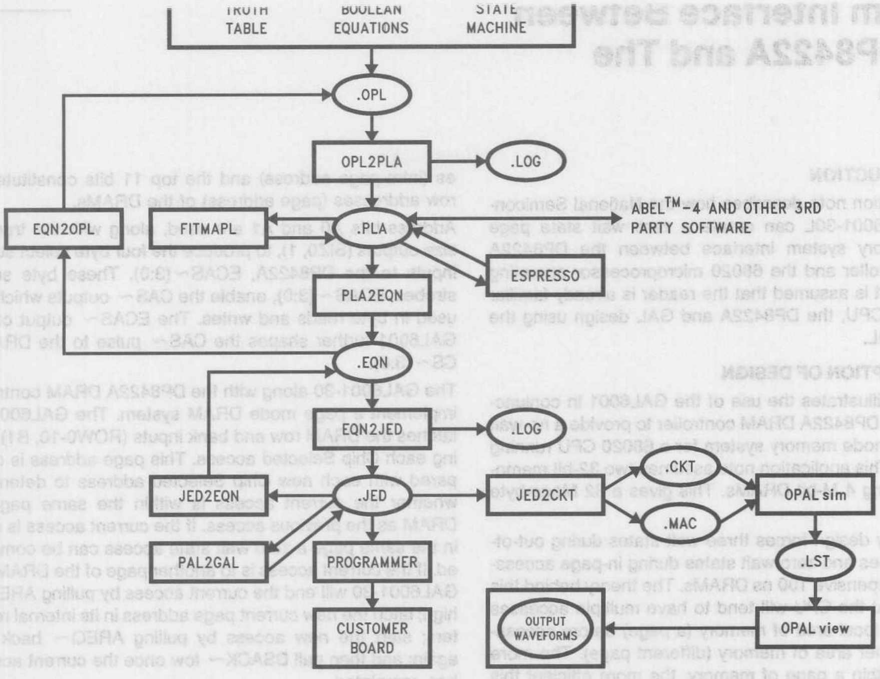
6.0 PAL TO GAL CROSS REFERENCE

PAL NSID	I_{CC} (mA)	t_{pd} (ns)	GAL NSID	I_{CC} (mA)	t_{pd} (ns)	GAL QS NSID
PAL10H8/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL10L8/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL12H6/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL12L6/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL14H4/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL14L4/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16H2/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16L2/A	90	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16C1/A	90	35/25	No Equivalent			
PAL10H8A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL10L8A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL12H6A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL12L6A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL14H4A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL14L4A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL16H2A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL16L2A2	45	35	GAL16V8-25Q	55	25	GAL16V8QS-25Q
PAL16C1A2	45	35	No Equivalent			
PAL16L8/A	180	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R4/A	180	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R6/A	180	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R8/A	180	35/25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16L8A2	90	35	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R4A2	90	35	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R6A2	90	35	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R8A2	90	35	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16L8B	180	15	GAL16V8-15L	90	15	GAL16V8QS-15L
PAL16R4B	180	15	GAL16V8-15L	90	15	GAL16V8QS-15L
PAL16R6B	180	15	GAL16V8-15L	90	15	GAL16V8QS-15L
PAL16R8B	180	15	GAL16V8-15L	90	15	GAL16V8QS-15L
PAL16L8B2	90	25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R4B2	90	25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R6B2	90	25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16R8B2	90	25	GAL16V8-25L	90	25	GAL16V8QS-25L
PAL16L8D	180	10	GAL16V8-10L	115	10	
PAL16R4D	180	10	GAL16V8-10L	115	10	
PAL16R6D	180	10	GAL16V8-10L	115	10	
PAL16R8D	180	10	GAL16V8-10L	115	10	
PAL16L8-7	180	7	GAL16V8-7L	115	7.5	
PAL16R4-7	180	7	GAL16V8-7L	115	7.5	
PAL16R6-7	180	7	GAL16V8-7L	115	7.5	
PAL16R8-7	180	7	GAL16V8-7L	115	7.5	

6.0 PAL TO GAL CROSS REFERENCE (Continued)

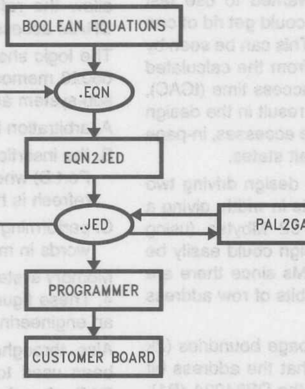
PAL NSID	I_{CC} (mA)	t_{pd} (ns)	GAL NSID	I_{CC} (mA)	t_{pd} (ns)	GAL QS NSID
PAL14L8/A	100	40/25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL16L6/A	100	40/25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL18L4/A	100	40/25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL20L2/A	100	40/25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL20C1/A	100	40/25	No Equivalent			
PAL12L10/A	100	40/25	GAL22V10-25L	130	25	
PAL20L10/A	165	50/30	GAL22V10-25L	130	25	
PAL20X4/A	180	50/30	GAL22V10-25L	130	25*	
PAL20X8/A	180	50/30	GAL22V10-25L	130	25*	
PAL20X10/A	180	50/30	GAL22V10-25L	130	25*	
PAL20L8A	210	25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL20R4A	210	25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL20R6A	210	25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL20R8A	210	25	GAL20V8-25L	90	25	GAL20V8QS-25L
PAL20L8B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20R4B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20R6B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20R8B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20P8B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20RP4B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20RP6B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20RP8B	210	15	GAL20V8A-15L	90	15	GAL20V8QS-15L
PAL20L8D	210	10	GAL20V8-10L	115	10	
PAL20R4D	210	10	GAL20V8-10L	115	10	
PAL20R6D	210	10	GAL20V8-10L	115	10	
PAL20R8D	210	10	GAL20V8-10L	115	10	
PAL20L8-7	210	7	GAL20V8-7L	115	7.5	
PAL20R4-7	210	7	GAL20V8-7L	115	7.5	
PAL20R6-7	210	7	GAL20V8-7L	115	7.5	
PAL20R8-7	210	7	GAL20V8-7L	115	7.5	
PAL16RA8	170	35	No Equivalent			
PAL20RA10	200	35	GAL20RA10-25	100	25	

*GAL22V10 can replace 90% of designs for PAL20X4/20X8/20X10 by conversion using OPAL software.



TL/L/11293–12

OPALjr Software Flow



TL/L/11293–13

FIGURE 12. OPAL and OPALjr software flows

A GAL6001-30L Zero Wait State Page Mode Memory System Interface Between The DP8422A and The 68020

National Semiconductor
Application Note 667

1.0 INTRODUCTION

This application note describes how the National Semiconductor GAL6001-30L can create a zero wait state page mode memory system interface between the DP8422A DRAM controller and the 68020 microprocessor operating at 16 MHz. It is assumed that the reader is already familiar with 68020 CPU, the DP8422A and GAL design using the GAL6001-30L.

2.0 DESCRIPTION OF DESIGN

This design illustrates the use of the GAL6001 in conjunction with the DP8422A DRAM controller to provide a no-wait state page-mode memory system for a 68020 CPU running at 16 MHz. This application note assumes two 32-bit memory banks using 4 M-bit DRAMs. This gives a 32 Mega-byte memory.

This memory design forces three wait states during out-of-page accesses and zero wait states during in-page accesses using inexpensive 100 ns DRAMs. The theory behind this design is that the CPU will tend to have multiple accesses within some local area of memory (a page) before accessing some other area of memory (different page). The more accesses within a page of memory, the more efficient this memory design allows the CPU to become. The page size of a 4 M-bit DRAM is 2048 bits. The page size of one bank of memory (32 bits per bank) is 8192 bytes or 8 Kbytes.

It should be noticed that if the user wanted to use fast DRAMs (80 ns or less access times) he could get rid of one wait state during out-of-page accesses. This can be seen by subtracting one clock period (62.5 ns) from the calculated RAS access time (tRAC) and the CAS access time (tCAC), section IV numbers 5 and 6. This would result in the design forcing two wait states during out-of-page accesses, in-page accesses would still remain with zero wait states.

Figure 1 shows a block diagram of this design driving two banks of DRAM, each bank being 32 bits in width, giving a maximum memory capacity of up to 32 Mbytes (using 4 M-bit × 1 DRAMs). This memory design could easily be changed to four banks of 1 M-bit DRAMs since there are 12 bits that are compared internally, 10 bits of row address for 1 M-bit DRAMs and 2 bank bits.

The memory banks are interleaved on page boundries (2k double word boundaries). This means that the address bit (A13) is tied to the bank select input of the DP8422A (B1). The bottom 11 bits (A2–12) constitute the column addresses (intra-page address) and the top 11 bits constitute the row addresses (page address) of the DRAMs.

Address bits A0 and A1 are used, along with the transfer size outputs (SIZ0, 1), to produce the four byte select strobe inputs to the DP8422A, ECAS~(3:0). These byte select strobes, ECAS~(3:0), enable the CAS~ outputs which are used in byte reads and writes. The ECAS~ output of the GAL6001 further shapes the CAS~ pulse to the DRAMs, CS~(3:0).

The GAL6001-30 along with the DP8422A DRAM controller implement a page mode DRAM system. The GAL6001-30 latches the DRAM row and bank inputs (ROW0-10, B1) during each Chip Selected access. This page address is compared with each new Chip Selected address to determine whether the current access is within the same page of DRAM as the previous access. If the current access is within the same page a zero wait state access can be completed. If the current access is to another page of the DRAM the GAL6001-30 will end the current access by pulling AREQ~ high; latch the new current page address in its internal registers; start the new access by pulling AREQ~ back low again; and then pull DSACK~ low once the current access has completed.

If AS~ from the 68020 is high and a refresh is requested (RFRQ~ low) the GAL6001-30 will end the current page mode access by pulling AREQ~ high. Then the GAL will allow the refresh to take place and start the next CPU DRAM access if one has been requested.

The logic shown in this application note forms a complete 68020 memory sub-system, no other logic is needed. This sub-system automatically takes care of:

A. arbitration between Port A and refreshing the DRAM;

B. the insertion of wait states to the processor (Port A and Port B) when needed (i.e., if RAS~ precharge is needed, refresh is happening during a memory access . . . etc.);

C. performing byte writes and reads to the 32-bit double words in memory.

Memory system timing diagrams appear in *Figures 2, 3,* and *4.* These figures are the result of simulating this design on an engineering workstation.

Also, throughout this application note the symbol "~" has been used to denote an active low signal. For example RAS~0 refers to the active low RAS0 output of the DP8421A.

3.0 DP8422A PROGRAMMING MODE BITS

Programming Bits	Description
R0 = 0	RAS~ low two clocks, RAS~ precharge of two clocks. If more RAS~ precharge is desired the user should program three periods of RAS~ precharge
R1 = 1	
R2 = 1	DTACK~ 1 is chosen. DTACK~ follows the access RAS~ low on the following rising clock edge
R3 = 0	
R4 = 0	No WAIT states during burst accesses
R5 = 0	
R6 = 0	If WAITIN~ = 0, add one clock to DTACK~. WAITIN~ may be tied high or low in this application depending upon the number of wait states the user desires to insert into the access
R7 = 1	Select DTACK~
R8 = 1	Non-interleaved Mode
R9 = X	
C0 = 0	Select based upon the input "DELCLK" frequency. Example: if the input clock frequency is 16 MHz then choose C0, 1, 2 = 0, 1, 0 (divide by eight, this will give a frequency of 2 MHz).
C1 = 1	
C2 = 0	
C3 = X	
C4 = 0	RAS banks selected by "B1". This mode allows two RAS~ outputs to go low during an access, and allows byte writing in 16-bit words.
C5 = 0	
C6 = 1	
C7 = 1	Column address setup time of 0 ns.
C8 = 1	Row address hold time of 15 ns
C9 = 1	Delay CAS~ during write accesses to one clock after RAS~ transitions low
B0 = 1	Fall through latches.
B1 = 1	Access mode 1
ECAS~0 = 1	Allow CAS~ to be extended after RAS~ transitions high. Also, allow the WE~ output to be used as a refresh request (RFRQ~) output indicator.

0 = Program with low voltage level

1 = Program with high voltage level

X = Program with either high or low voltage level (don't care condition)

4.0 16 MHz 68020 TIMING CALCULATIONS FOR A SYSTEM WITH THREE WAIT STATES DURING NORMAL ACCESSES AND ZERO WAIT STATES DURING BURST ACCESSES

1. Maximum time to CS~ valid:
   30 ns (68020RC16 max time to valid address)

2. Minimum time to ADS~ valid:
   62.5 ns (one clock period at 16 MHz)
   +4 ns (GAL6001-30 assumed min time output clock to AREQ~ valid)
   = 66.5 ns

3. Minimum CS~ setup time to ADS~ valid (DP8422A-25 needs a minimum of 5 ns):
   66.5 ns (see #2 above)
   −30 ns (see #1 above)
   = 36.5 ns

4. Minimum CS~ setup time to CLOCK high (GAL6001-30 needs 25 ns input setup time to the output CLOCK for the AREQ~ output):
   62.5 ns (one clock)
   −30 ns (max time to address bit 31 valid, see #1 above)
   = 32.5 ns

5. Determining tRAC during a normal access (RAS~ access time needed by the DRAM):
   217.5 ns (three and one half clocks, (3 × 62.5) + 30 = 217.5 ns)
   −15 (GAL6001-30 max CLK to AREQ~ valid)
   −29 ns (ADS~ to RAS~ low max, DP8422A-25 #402)
   −7 ns (74F245 max delay)
   −5 ns (68020 data setup time)
   = 161.5 ns
   Therefore the tRAC of the DRAM must be 161.5 ns or less.

6. Determining tCAC during a normal access (CAS~ access time)
   217.5 ns (three and one half clocks, (3 × 62.5) + 30 = 217.5 ns)
   −15 (GAL6001-30 max CLK to AREQ~ valid)
   −75 ns (ADS~ to CAS~ low max, DP8422A-25 #403a, light load)
   −14 ns (74F32 CS~(3:0) drivers max delay driving 125 pF)
   −7 ns (74F245 max delay)
   −5 ns (68020 data setup time)
   = 102.5 ns
   Therefore the tCAC and the column address access time of the DRAM must be 102.5 ns or less.

7. Maximum time to CS~(3:0) low during a page mode access:
   62.5 ns (one clock at 16 MHz)
   +30 ns (GAL6001-30 max time from clock to output, ECAS~)
   +14 ns (74F32 max time to CS~(3:0) valid)
   = 106.5 ns

8. Minimum time to DRAM column address strobes low [CS~(3:0)] during a page mode access:
   62.5 ns (one clock at 16 MHz)
   +8 ns (assumed GAL6001-30 min time from input to output, ECAS~)
   +4 ns (assumed 74F32 min time to CS~(3:0) valid)
   = 74.5 ns

9. Determining the minimum column address setup time to CS~(3:0) low (0 ns needed by the DRAMs) during burst mode accesses for zero wait states:
   74.5 ns (see #8 above, min time to CS~(3:0) valid)
   −30 ns (max time to 68020 address valid)
   −35 ns (DP8422A-25 max time address in to out, #27)
   = 9.5 ns minimum
10. Determining the tCAC (CAS~ access time) needed during burst mode accesses for zero wait states:
    155 ns (two and one half clocks, (2 × 62.5) + 30 = 155 ns)
    −106.5 ns (max time to CS~(3:0), see #7 above)
    −7 ns (74F245 max delay)
    −5 ns (68020 data setup time)
    = 36.5 ns
11. Determining the column address access time needed during burst mode accesses for zero wait states:
    155 ns (two and one half clocks, (2 × 62.5) + 30 = 155 ns)
    −30 ns (max time to 68020 address valid)
    −35 ns (DP8422A-25 max time address in to out, #27)
    −7 ns (74F245 max delay)
    −5 ns (68020 data setup time)
    = 78 ns
12. Minimum DSACK~ (Data transfer and Size ACKnowledge) setup time to clock low (68020 DSACK~ input needs 5 ns, #47a) during page mode zero wait state accesses:
    30 ns (one half clock period, S2 clock of 68020 clock cycle)
    −25 ns (GAL6001-30 input to outputs enabled, DSACK~ output)
    = 5 ns

Note: Calculations can be performed for different frequencies, different logic (ALS or CMOS . . . etc.), and/or different combinations of wait states by **substituting** the appropriate values into the above equations.

5.0 68020 GAL6001-30 INPUT AND OUTPUT DESCRIPTIONS

Inputs:

ROW0–10 These are the row address inputs of the DRAMs and are also connected to the R0–10 inputs of the DP8422A-25. The GAL6001-30 latches these inputs along with the B1 input and compares this address with each new address during a Chip Selected DRAM access to determine whether the current access is within the same page of DRAM as the previous access.

B1 The bank input to the DP8422A-25, B1 input. This input determines which of the two DRAM banks the CPU is currently accessing in. The GAL6001-30 latches this input along with the ROW0–10 inputs and compares this address with each new address during a Chip Selected DRAM access to determine whether the current access is within the same page of DRAM as the previous access.

RFRQ~ The ReFresh ReQuest input from the DP8422A DRAM controller.

READ The 68020 READ and write access indicator.

DTACK~ The DP8422A Data Transfer ACKnowledge indicator.

AS~ The 68020 address strobe, indicating that the CPU address is valid and a CPU access is in progress.

CS~ Chip Select for the memory system. It was assumed that the 68020 address bit 31 would be used for this indicator. When low it indicates that the 68020 is accessing the DRAM.

CLK, ICLK The 68020 system clock, 16 MHz in this application.

Outputs:

AREQ~ The DRAM Access REQuest. This signal is input to the DP8422A DRAM controller. It will remain low as long as all 68020 chip selected accesses remain within the current page. As soon as an access occurs that is not within the currently latched page address or a refresh request occurs AREQ~ will be pulled high.

ECAS~ Enable CAS~ is toggled during every access and is used to drive the CAS~ inputs to the DRAMs, CS~(3:0). This input is delayed during write accesses to allow time for the data to become valid at the DRAM inputs before CAS~ transitions low. The READ input to the DRAMs is guaranteed to transition while ECAS~ is high.

DSACK~ The Data transfer and Size ACKnowledge output goes to the 68020 to end the current access when low.

Internal Nodes:

LR0–10 These are the latched ROW0–10 addresses of the current page of DRAM. These addresses are clocked by the falling edge of CS_AS_L~.

LB1 This is the latched B1 address of the current page of DRAM.

CS_AS_L~ This is a latched version of Chip Select and Address Strobe of the 68020. This signal toggles during each access and transitions low from the rising edge of S2 clock and high from the rising edge of S5 clock.

AS_D2~ This is a delayed version of CS_AS_L~.

RFRQD~ This is the DP8422A ReFresh ReQuest Delayed and Synchronized to the 68020 system clock.

6.0 68020 GAL6001-30 EQUATIONS WRITTEN IN NATIONAL SEMICONDUCTOR OPAL FORMAT

```
TITLE      68020/DP8422A DRAM PAGE DETECTOR FOR USE WITH NATIONAL GAL6001
PATTERN    PG_DETECT
REVISION   A
AUTHOR     RUSTY MEIER
COMPANY    NATIONAL SEMICONDUCTOR
DATE       DEC. 11, 1989
CHIP       PG_DETECT     GAL6001
;PIN LIST
R0 R1 R2 R3 R4 R5 R6 R7 RFRQ~ DTACK~ AS~ GND
CLK R8 R9 R10 B1 CS~ ICLK DSACK~ AREQ~ ECAS~ READ VCC
;BURIED NODE OUTPUTS
LR0 LR1 LR2 LR3 LR4 LR5 LR6 LR7
;DUAL FEEDBACK NODE OUTPUTS
LR8 LR9 LR10 LB1 CS_AS_L~ AS_D2~ NC NC NC RFRQD~
EQUATIONS
;GAL DUAL FEEDBACK NODES ****************************************
;NOTICE THAT THE CLOCKS (XXX.CLKF TERMS)
;ARE THE SAME AS "CS_AS_L~" INVERTED
LR8 := R8
LR8.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR9 := R9
LR9.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR10 := R10
LR10.CLKF  = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LB1 := B1
LB1.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
!CS_AS_L~  = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
!AS_D2~    = !CS~ & !AS~ & !CS_AS_L~ & !ICLK
           # !CS~ & !AS~ & !AS_D2~
           # !CS~ & !AS_D2~ & !ICLK
!RFRQD~ := !RFRQ~
;GAL OUTPUTS ****************************************
DSACK~ = CS~
           # !CS~ & R0 & !LR0
           # !CS~ & !R0 & LR0
           # !CS~ & R1 & !LR1
           # !CS~ & !R1 & LR1
           # !CS~ & R2 & !LR2
           # !CS~ & !R2 & LR2
           # !CS~ & R3 & !LR3
```

```
                # !CS~ & !R5 & LR5
                # !CS~ & R6 & !LR6
                # !CS~ & !R6 & LR6
                # !CS~ & R7 & !LR7
                # !CS~ & !R7 & LR7
                # !CS~ & R8 & !LR8
                # !CS~ & !R8 & LR8
                # !CS~ & R9 & !LR9
                # !CS~ & !R9 & LR9
                # !CS~ & R10 & !LR10
                # !CS~ & !R10 & LR10
                # !CS~ & B1 & !LB1
                # !CS~ & !B1 & LB1
                # DTACK~
                # DSACK~ & ICLK & !AS_D2~
                # AS~ & !AS_D2~
                # AREQ~
DSACK~.TRST = !CS~ & !AS~
ECAS~ = CS~
                # !CS~ & R0 & !LR0
                # !CS~ & !R0 & LR0
                # !CS~ & R1 & !LR1
                # !CS~ & !R1 & LR1
                # !CS~ & R2 & !LR2
                # !CS~ & !R2 & LR2
                # !CS~ & R3 & !LR3
                # !CS~ & !R3 & LR3
                # !CS~ & R4 & !LR4
                # !CS~ & !R4 & LR4
                # !CS~ & R5 & !LR5
                # !CS~ & !R5 & LR5
                # !CS~ & R6 & !LR6
                # !CS~ & !R6 & LR6
                # !CS~ & R7 & !LR7
                # !CS~ & !R7 & LR7
                # !CS~ & R8 & !LR8
                # !CS~ & !R8 & LR8
                # !CS~ & R9 & !LR9
                # !CS~ & !R9 & LR9
                # !CS~ & R10 & !LR10
                # !CS~ & !R10 & LR10
                # !CS~ & B1 & !LB1
                # !CS~ & !B1 & LB1
                # AS~
                # CS~
                # ECAS~ & !ICLK & CS_AS_L~
                # AREQ~
                # !READ & ECAS~ & AS_D2~ & ICLK
                # !RFRQD~ & CS_AS_L~
AREQ~ :=          !CS~ & R0 & !LR0
                # !CS~ & !R0 & LR0
                # !CS~ & R1 & !LR1
                # !CS~ & !R1 & LR1
                # !CS~ & R2 & !LR2
```

```
          # !CS~ & R5 & !LR5
          # !CS~ & !R5 & LR5
          # !CS~ & R6 & !LR6
          # !CS~ & !R6 & LR6
          # !CS~ & R7 & !LR7
          # !CS~ & !R7 & LR7
          # !CS~ & R8 & !LR8
          # !CS~ & !R8 & LR8
          # !CS~ & R9 & !LR9
          # !CS~ & !R9 & LR9
          # !CS~ & R10 & !LR10
          # !CS~ & !R10 & LR10
          # !CS~ & B1 & !LB1
          # !CS~ & !B1 & LB1
          # !RFRQD~ & CS_AS_L~
          # AREQ~ & CS_AS_L~
;BURIED NODES ****************************************
; NOTICE THAT THE CLOCKS (xxx.CLKF TERMS) ARE THE
; SAME AS 'CS_AS_L~' INVERTED
LR0 := R0
LR0.CLKF   = !CS~ & !AS~ & ICLK
           = !CS_AS_L~ & !AS~
           = !CS_AS_L~ & !ICLK
LR1 := R1
LR1.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR2 := R2
LR2.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR3 := R3
LR3.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR4 := R4
LR4.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR5 := R5
LR5.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR6 := R6
LR6.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
LR7 := R7
LR7.CLKF   = !CS~ & !AS~ & ICLK
           # !CS_AS_L~ & !AS~
           # !CS_AS_L~ & !ICLK
```

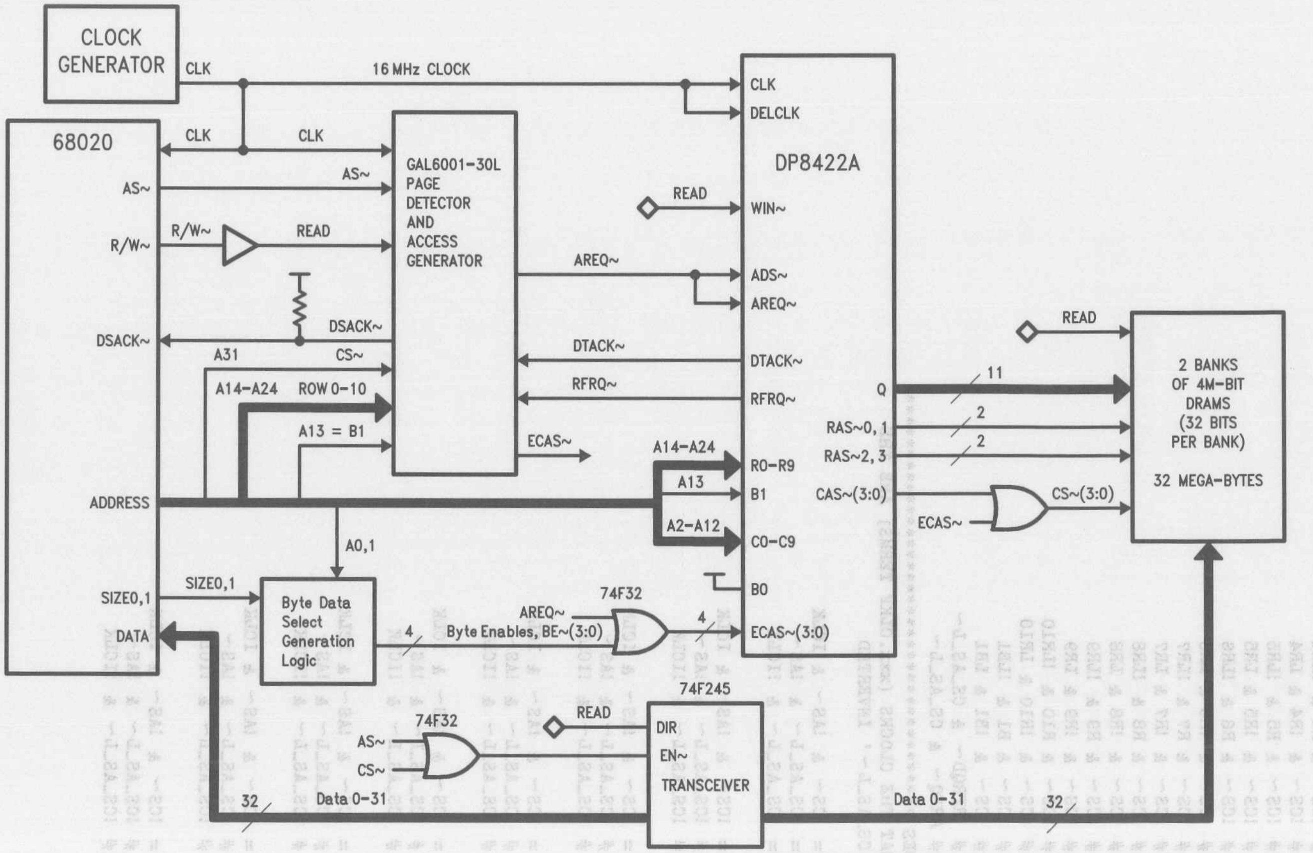
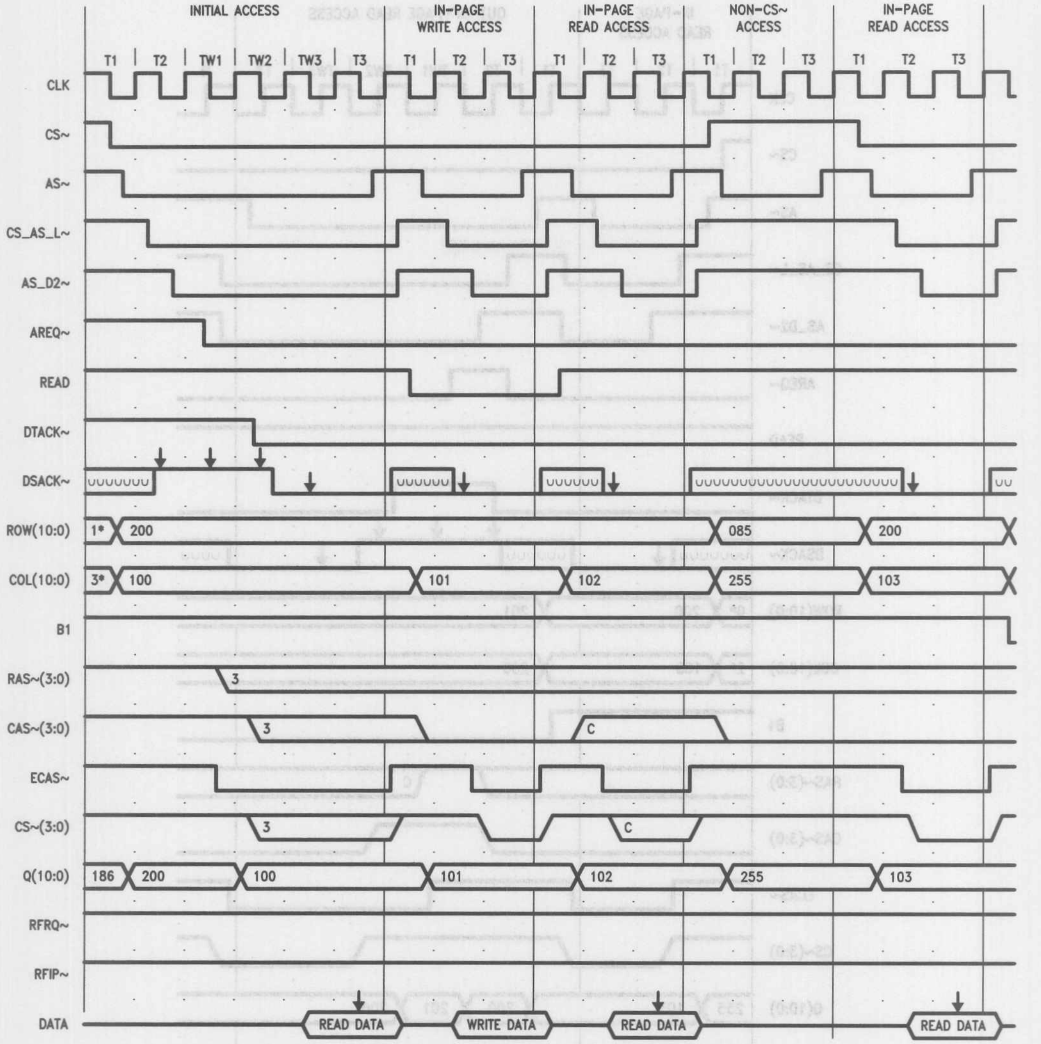
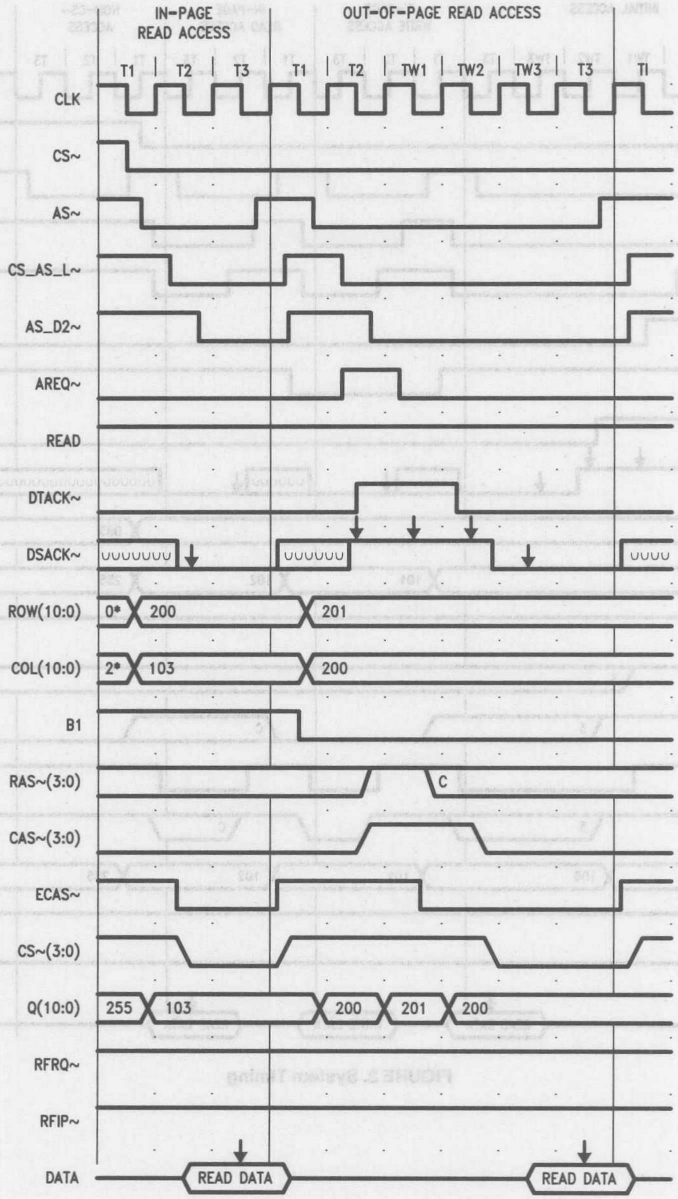


FIGURE 1. A GAL6001 Interface to the 68020/DP8422A-25/DRAM Using Page Mode Accessing



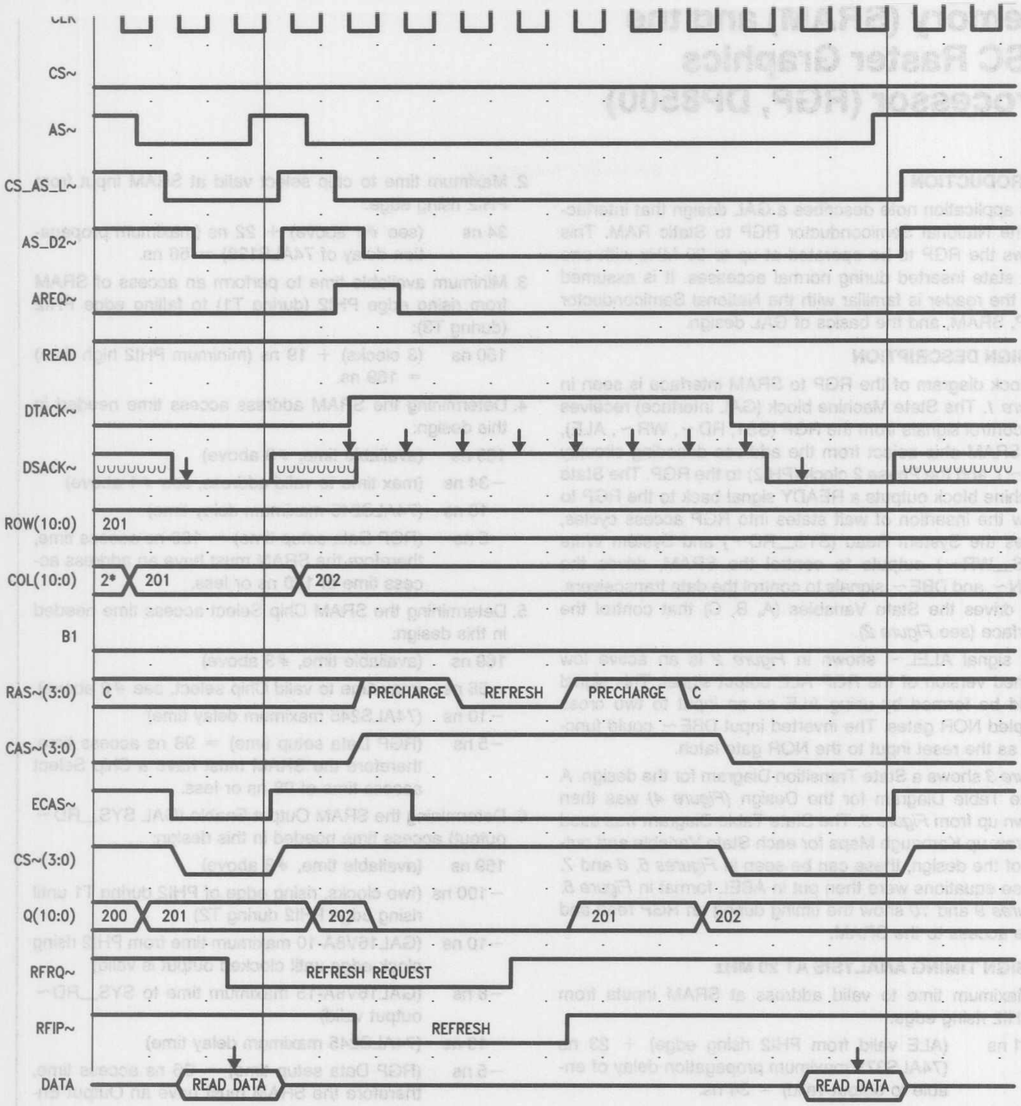
TL/L/10771-5

FIGURE 2. System Timing



TL/L/10771-6

FIGURE 3. System Timing



TL/L/10771-7

FIGURE 4. System Timing

A GAL Interface between Static Random Access Memory (SRAM) and the NSC Raster Graphics Processor (RGP, DP8500)

National Semiconductor
Application Note 669

INTRODUCTION

This application note describes a GAL design that interfaces the National Semiconductor RGP to Static RAM. This allows the RGP to be operated at up to 20 MHz with one wait state inserted during normal accesses. It is assumed that the reader is familiar with the National Semiconductor RGP, SRAM, and the basics of GAL design.

DESIGN DESCRIPTION

A block diagram of the RGP to SRAM interface is seen in *Figure 1.* The State Machine block (GAL interface) receives the control signals from the RGP (BS1, RD ~ , WR ~ , ALE), the SRAM chip select from the address decoding circuitry (CS ~), and the Phase 2 clock (PHI2) to the RGP. The State Machine block outputs a READY signal back to the RGP to allow the insertion of wait states into RGP access cycles, drives the System Read (SYS_RD ~) and System Write (SYS_WR ~) outputs to control the SRAM, drives the DDIN ~ and DBE ~ signals to control the data transceivers, and drives the State Variables (A, B, C) that control the interface (see *Figure 2*).

The signal ALEL ~ shown in *Figure 2* is an active low latched version of the RGP ALE output signal. This signal could be formed by using ALE as an input to two cross coupled NOR gates. The inverted input DBE ~ could function as the reset input to the NOR gate latch.

Figure 3 shows a State Transition Diagram for the design. A State Table Diagram for the Design *(Figure 4)* was then drawn up from *Figure 3.* The State Table Diagram was used to draw up Karnaugh Maps for each State Variable and output of the design, these can be seen in *Figures 5, 6* and *7.* These equations were then put in ABEL format in *Figure 8. Figures 9* and *10* show the timing during an RGP read and write access to the SRAM.

DESIGN TIMING ANALYSIS AT 20 MHz

1. Maximum time to valid address at SRAM inputs from PHI2 rising edge:

   11 ns (ALE valid from PHI2 rising edge) + 23 ns (74ALS373 maximum propagation delay of enable to output valid) = 34 ns.

2. Maximum time to chip select valid at SRAM input from PHI2 rising edge:

   34 ns (see #1 above) + 22 ns (maximum propagation delay of 74ALS138) = 56 ns.

3. Minimum available time to perform an access of SRAM from rising edge PHI2 (during T1) to falling edge PHI2 (during T3):

   150 ns (3 clocks) + 19 ns (minimum PHI2 high time) = 169 ns.

4. Determining the SRAM address access time needed in this design:

   169 ns (available time, #3 above)
   −34 ns (max time to valid address, see #1 above)
   −10 ns (74ALS245 maximum delay time)
   −5 ns (RGP Data setup time) = 120 ns access time, therefore the SRAM must have an address access time of 120 ns or less.

5. Determining the SRAM Chip Select access time needed in this design:

   169 ns (available time, #3 above)
   −56 ns (max time to valid Chip select, see #2 above)
   −10 ns (74ALS245 maximum delay time)
   −5 ns (RGP Data setup time) = 98 ns access time, therefore the SRAM must have a Chip Select access time of 98 ns or less.

6. Determining the SRAM Output Enable (GAL SYS_RD ~ output) access time needed in this design:

   169 ns (available time, #3 above)
   −100 ns (two clocks, rising edge of PHI2 during T1 until rising edge PHI2 during T2)
   −10 ns (GAL16V8A-10 maximum time from PHI2 rising clock edge until clocked output is valid)
   −8 ns (GAL16V8A-15 maximum time to SYS_RD ~ output valid)
   −10 ns (74ALS245 maximum delay time)
   −5 ns (RGP Data setup time) = 36 ns access time, therefore the SRAM must have an Output Enable access time of 36 ns or less.

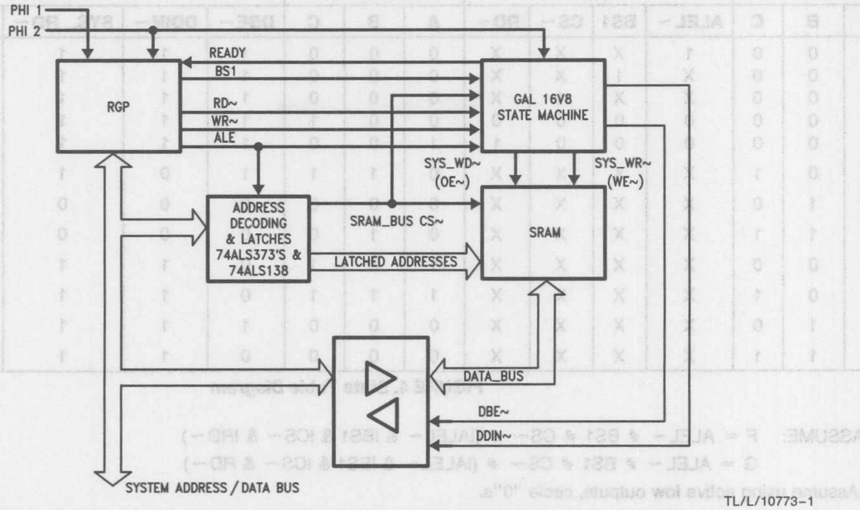


FIGURE 1. Block Diagram of Raster Graphics Processor (RGP) to Static Random Access Memory (SRAM) Interface

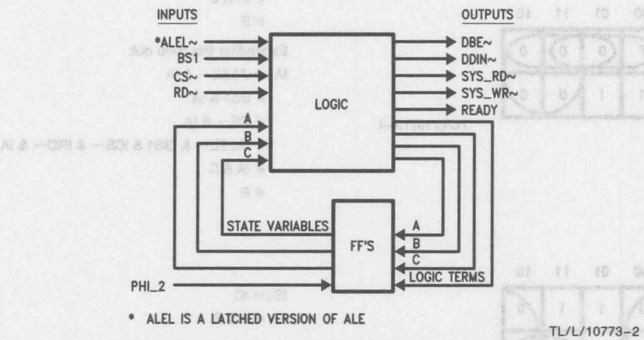


FIGURE 2. Synchronized State Machine Model

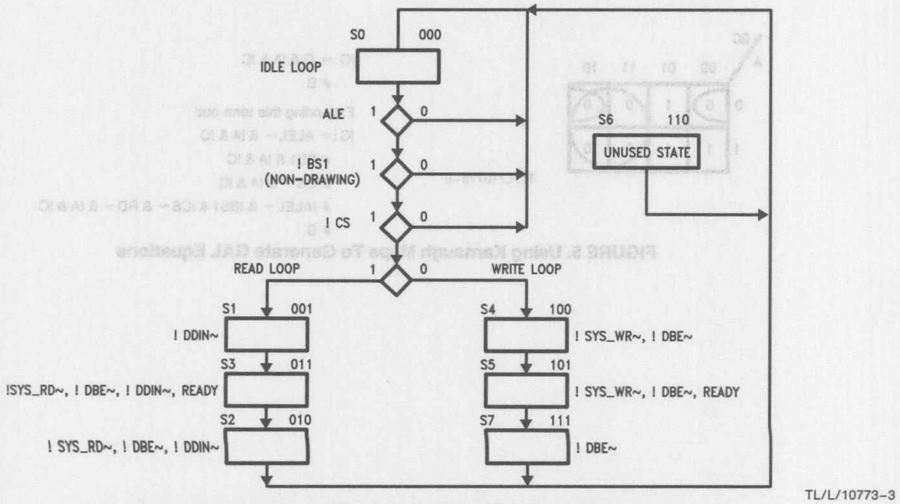


FIGURE 3. State Transition Diagram for RGP/SRAM Interface Design

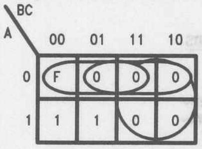
Present State			Inputs				Next State			Outputs				
A	B	C	ALEL~	BS1	CS~	RD~	A	B	C	DBE~	DDIN~	SYS_RD~	SYS_WR~	READY
0	0	0	1	X	X	X	0	0	0	1	1	1	1	0
0	0	0	X	1	X	X	0	0	0	1	1	1	1	0
0	0	0	X	X	1	X	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
0	0	0	0	0	0	1	1	0	0	1	1	1	1	0
0	0	1	X	X	X	X	0	1	1	1	0	1	1	0
0	1	0	X	X	X	X	0	0	0	0	0	0	1	0
0	1	1	X	X	X	X	0	1	0	0	0	0	1	1
1	0	0	X	X	X	X	1	0	1	0	1	1	0	0
1	0	1	X	X	X	X	1	1	1	0	1	1	0	1
1	1	0	X	X	X	X	0	0	0	1	1	1	1	0
1	1	1	X	X	X	X	0	0	0	0	1	1	1	0

FIGURE 4. State Table Diagram

*ASSUME: F = ALEL~ # BS1 # CS~ # (!ALEL~ & !BS1 & !CS~ & !RD~)

G = ALEL~ # BS1 # CS~ # (!ALEL~ & !BS1 & !CS~ & RD~)

* Assume using active low outputs, circle "0"s.

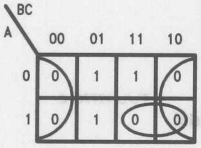


TL/L/10773-4

!A := F & !A
!A & C
B

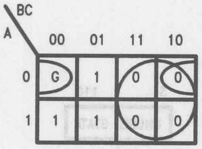
Expanding this term out:

!A := ALEL~ & !A
# BS1 & !A
# CS~ & !A
!ALEL~ & !BS1 & !CS~ & !RD~ & !A
!A & C
B



TL/L/10773-5

!B := !C
A & B



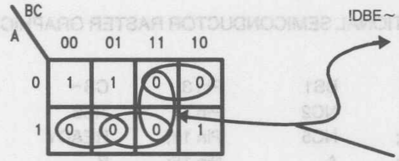
TL/L/10773-6

!C := G & !A & !C
B

Expanding this term out:

!C := ALEL~ & !A & !C
# BS1 & !A & !C
CS~ & !A & !C
!ALEL~ & !BS1 & !CS~ & RD~ & !A & !C
B

FIGURE 5. Using Karnaugh Maps To Generate GAL Equations

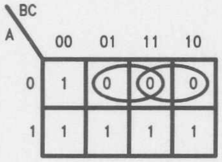


	00	01	11	10
0	1	1	0	0
1	0	0	0	1

TL/L/10773–7

!DBE~ = A & !B
B & C
!A & B
A & C

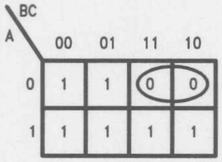
This term is not needed in this example because there never is a transition between states '011' and '111'.



	00	01	11	10
0	1	0	0	0
1	1	1	1	1

!DDIN~ = !A & C
!A & B

TL/L/10773–8

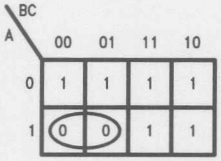


	00	01	11	10
0	1	1	0	0
1	1	1	1	1

!SYS_RD~ = !A & B

TL/L/10773–9

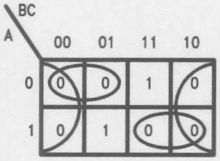
FIGURE 6. Using Karnaugh Maps To Generate GAL Equations



	00	01	11	10
0	1	1	1	1
1	0	0	1	1

!SYS_WR~ = A & !B

TL/L/10773–10



	00	01	11	10
0	0	0	1	0
1	0	1	0	0

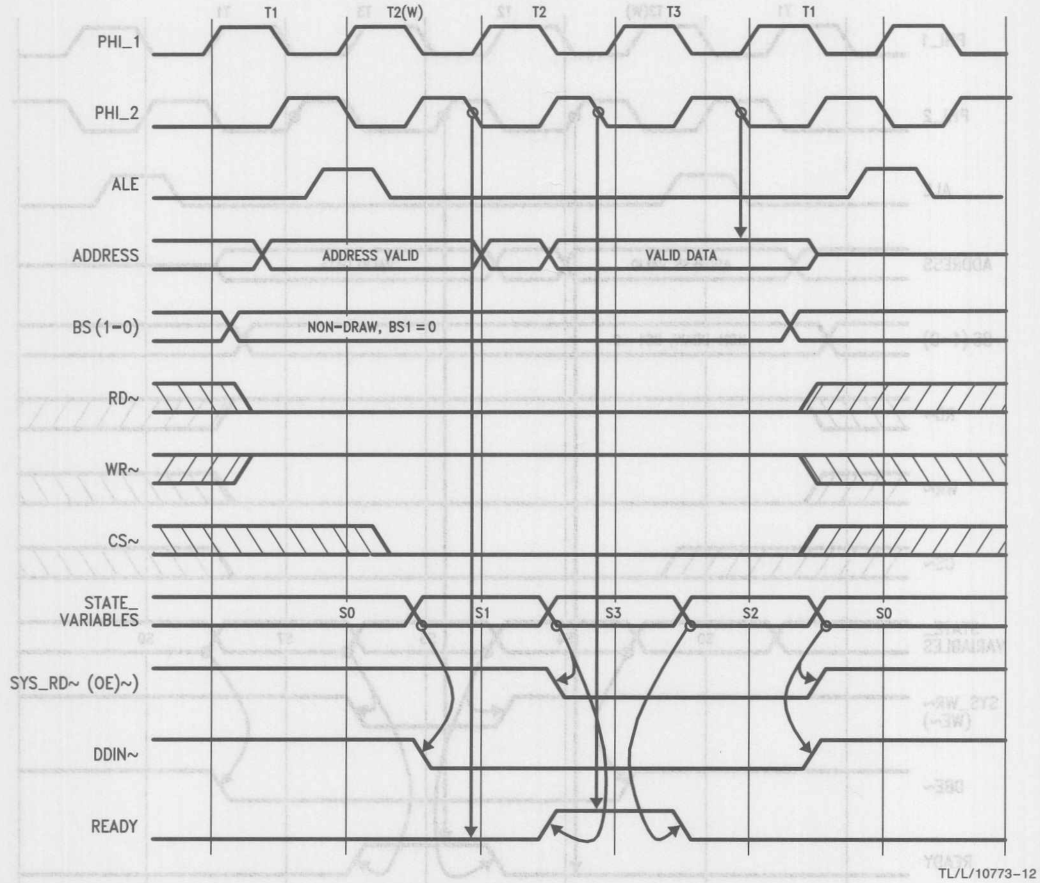
!READY = !A & !B
A & B
!C

TL/L/10773–11

FIGURE 7. Using Karnaugh Maps To Generate GAL Equations

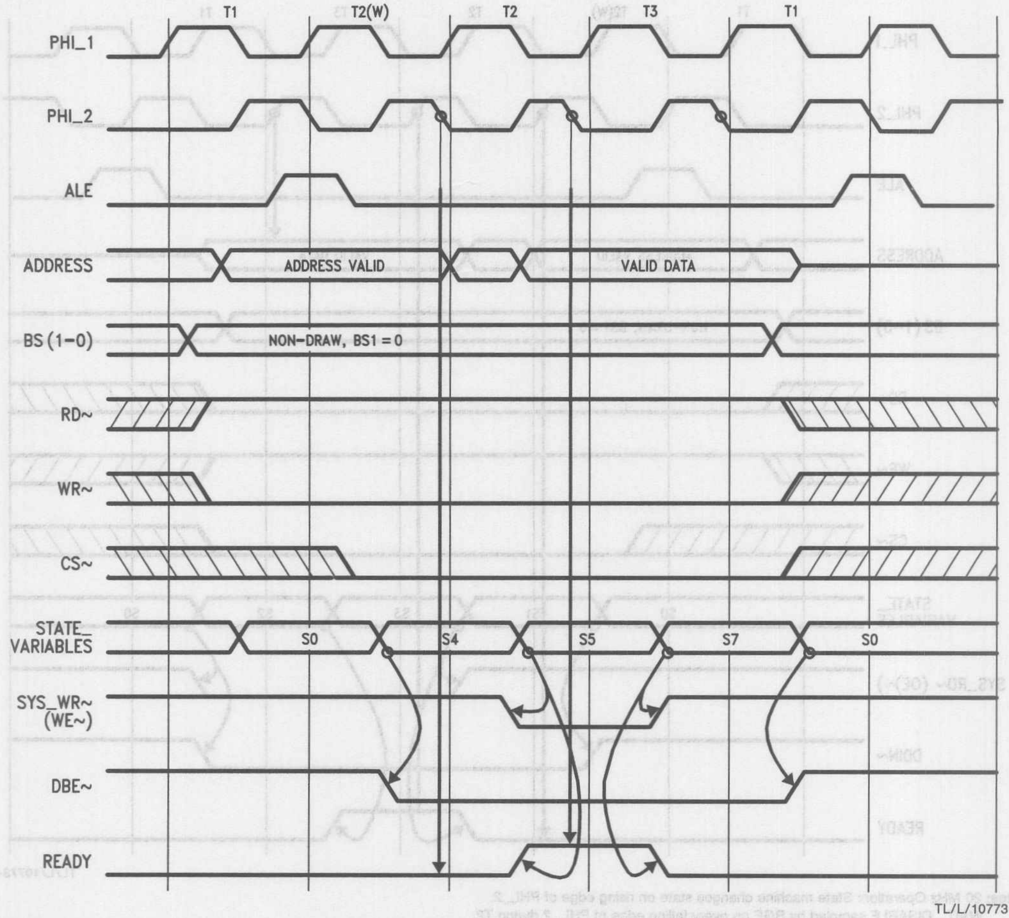
```
MODULE SRAM__INTERFACE
TITLE     'SRAM__GAL, THIS GAL INTERFACES THE NATIONAL SEMICONDUCTOR RASTER GRAPHICS PROCESSOR TO
           A STATIC RANDOM ACCESS MEMORY'.
SRAM__GAL      device    16V8
PHI__2         Pin 1;    ALEL__        Pin 2;    BS1      Pin 3;    CS~       Pin 4;
RD~            Pin 5;    NC1           Pin 6;    NC2      Pin 7;    NC3       Pin 8;
NC4            Pin 9;    GND           Pin 10;   NC5      Pin 11;   READY     Pin 12;
SYS__RD~       Pin 13;   SYS__WR~      Pin 14;   A        Pin 15;   B         Pin 16;
C              Pin 17;   DDIN~         Pin 18;   DBE~     Pin 19;   VCC       Pin 20;

EQUATIONS

!A :           = ALEL~ & !A
               # BS1 & !A
               # CS~ & !A
               # !ALEL~ & !BS1 & !CS~ & !RD~ & !A
               # !A & C
               # B

!B :           = !C
               # A & B

               = ALEL~ & !A & !C
!C :           # BS1 & !A & !C
               # CS~ & !A & !C
               # !ALEL~ & !BS1 & !CS~ & RD~ & !A & !C
               # B

!DBE~          = A & !B
               # B & C
               # !A & B
               # A & C

!DDIN~         = !A & C
               # !A & B

!SYS__RD~      = !A & B

!SYS__WR~      = A & !B

!READY         = !A & !B
               # A & B
               # !C
```

FIGURE 8. ABEL GAL Equations



Notes: 20 MHz Operation: State machine changes state on rising edge of PHI__2.
WAIT__DISABLE sampled by RGP on every falling edge of PHI__2 during T2.
Data sampled by RGP on falling edge of PHI__2 during T3.

FIGURE 9. Non Draw SRAM Read Operation



Notes: 20 MHz Operation: State machine changes on rising edge of PHI__2.
Wait sampled on every falling edge of PHI__2 during T2.

FIGURE 10. Non Draw SRAM Write Operation

Mode DRAM Memory System

I. INTRODUCTION

This application note describes a two GAL designs that interface the DP8422A to the 80286 CPU. The first design allows the 80286 to be operated at up to 40 MHz (80286-20) with one wait state inserted during normal accesses. The second design allows the 80286 to operate at up to 40 MHz (80286-20) with zero wait states inserted when operating the DRAMs in page mode. Design number two also makes use of the 74ALS6311 page detector to determine whether the 80286 current access is within the same page as the previous access. It is assumed that the reader is familiar with the 80286, the DP8422A DRAM controller, the 74ALS6311 and the basics of GAL design.

II. DESCRIPTION OF DESIGN #1, 80286 OPERATING AT UP TO 40 MHz WITH ONE WAIT STATED (80286-20)

The block diagram of this design is shown driving two banks of DRAM, each bank being 16 bits in width, giving a maximum memory capacity of up to 4 Mbytes (using 1 Mbit x 1 DRAMs). This memory could easily be expanded up to 32 Mbytes using four banks of 4 Mbit DRAMs.

The memory banks are interleaved on word (16-bit word) boundaries. This means that the address bit (A1) is tied to the bank select input of the DP8422A (B1).

Address bit A0 is used, along with Bus High Enable (BHE), to produce the two byte select ECAS~0,1 strobes. These byte select strobes (ECAS~0,1) enable the CAS~ outputs which are used in byte reads and writes.

If the majority of accesses made by the 80286 are sequential, the 80286 can alternate memory banks, allowing one memory bank to be precharging (RAS~ precharge) while the other banks are being accessed. Each separate memory access to the same memory bank will require extra wait states to be inserted into the CPU access cycles to allow for the RAS~ precharge time.

This application inserts 1 wait state in normal accesses of the 80286. The number of wait states can be adjusted through the WAITIN input of the DP8422A.

The logic shown in this application note forms a complete 80286 memory sub-system, no other logic is needed. This sub-system automatically takes care of:

A. Arbitration between Port A, Port B, and refreshing the DRAM;

B. The insertion of wait states to the processor (Port A and Port B) when needed (i.e., if RAS~ precharge is needed, refresh is happening during a memory access, the other Port is currently doing an access . . . etc);

C. Performing byte writes and reads to the 16-bit words in memory.

It is important that the 74AS00 NAND gates (U1) be in the same package so these delays (CLK~, S01) track each other.

By using the "output control" pins of some external latches (74AS373's), this application can easily be used in a dual access application. The addresses could be tri-stated through these latches, the write input (WIN~), lock input (LOCK~), and ECAS~0–3 inputs must also be able to be tri-stated (a 74AS244 could be used for this purpose). By multiplexing the above inputs (through the use of the above parts and similar parts for Port B) the DP8422A can be used in a dual access application. If this design is used in a dual access application the tRAC and tCAC (required RAS and CAS access time required by the DRAM) will have to be recalculated since the time to RAS and CAS is longer for the dual access application (see TIMING section of this application note).

Also, throughout this application note the symbol '~' has been used to denote and active low signal. For example RAS~0 refers to the active low RAS0 output of the DP8421A.

III. DESCRIPTION OF DESIGN #2, 80286 OPERATING AT UP TO 40 MHz (80286-20) WITH ZERO WAIT STATES USING PAGE MODE DRAMs

This design is very similar with respect to design #1 except for the following differences.

The memory banks are interleaved on page (1024 word) boundaries. This means that the address bit (A11) is tied to the bank select input of the DP8421A (B1).

Address bit A0 is used, along with Bus High Enable (BHE), to produce the two byte select ECAS~0,1 strobes. These byte select strobes (ECAS~0,1) are logically "ORed" with the DP8421A CAS~ outputs to produce the byte selecting CAS~ inputs to the DRAMs.

If the majority of accesses made by the 80286 are sequential and within a page, the 80286 in conjunction with the page detector (74ALS6311) allow zero wait state accessing. Each in-page memory access is completed using page mode (toggling the CAS~ inputs).

As in design #1 it is important that the 74AS00 NAND gates (U1) be in the same package so the delays (CLK~, S01) track each other.

IV. 80286 DESIGNS #1 AND #2 PROGRAMMING MODE BITS

Programming Bits	Description
R0 = 0	RAS~ low two clocks, RAS~ precharge of two clocks. If more RAS~ precharge is desired the user should program three periods of RAS~ precharge.
R1 = 1	
R2 = 0	DTACK~ ½ is chosen.
R3 = 1	DTACK~ follows the access RAS~ low.
R4 = 0	No WAIT states during burst accesses.
R5 = 0	
R6 = 0	If WAITIN~ = 0, add one clock to DTACK~. WAITIN~ may be tied high or low in this application depending upon the number of wait states the user desires to insert into the access.
R7 = 1	Select DTACK~.
R8 = 1	Non-interleaved Mode.
R9 = X	
C0 = X	Select based upon the input "DELCLK" frequency. Example: if the input clock frequency is 16 MHz then choose C0, 1, 2 = 0, 1, 0 (divide by eight, this will give a frequency of 2 MHz).
C1 = X	
C2 = X	
C3 = X	
C4 = 1	RAS banks selected by "B1".
C5 = 0	This mode allows two RAS~ outputs to go low during an access, and allows byte writing in 16 bit words.
C6 = 1	
C7 = 1	Column address setup time of 0 ns.
C8 = 1	Row address hold time of 15 ns.
C9 = 1	Delay CAS~ during write accesses to one clock after RAS~ transitions low.
B0 = 1	Fall through latches.
B1 = 1	Access mode 1.
ECAS~0 = 1	Allow CAS~ to be extended after RAS~ transitions high. Also, allow the WE~ output to be used as a refresh request (RFRQ~) output indicator.

0 = Program with low voltage level
1 = Program with high voltage level
X = Program with either high or low voltage level (don't care condition)

V. 80286 TIMING CALCULATIONS FOR DESIGNS #1 AND #2 AT 32 MHz (80286-16) WITH ONE WAIT STATE DURING NORMAL ACCESSES AND ZERO WAIT STATES IN PAGE MODE ACCESSES (DESIGN #2 ONLY). THE WAITIN~ INPUT OF THE DP8422A SHOULD BE TIED LOW.

1. Minimum S01 high setup time to CLK~ high:

   31.25 ns (one clock period, 32 MHz) − 20 ns (maximum 80286 S0~, S1~ delay, #12a) − 1 ns (maximum skew between CLK~ and S0~, S1~ since both gates are in the same package) = 10.25 ns.

2. Maximum address valid time (with respect to CLK~ high during phase 1 in Ts):

   62.5 ns (two clocks 32 MHz) − 31 ns (80286 address valid delay from previous clock period, #15) + 1 ns (minimum CLK~ valid delay, 74AS00) = −1.25 ns (before CLK~ high phase 1 Ts).

3. Minimum address setup time to ADS~ low (DP8421A-25 needs 14 ns, #404):

   31.25 ns (one clock period) + 1.25 ns (from #2 calculation above) + 2 ns (minimum ADS~ valid delay from CLK~ high, beginning of phase 2 in Ts) = 34.5 ns address setup.

4. Minimum CS setup time to ADS~ low (DP8421A-25 needs 5 ns, #401): 34.5 ns (#3 above) − 10 ns (max 74ALS138 decoder) = 24.5 ns.

5. Determining tRAC during a normal access (RAS~ access time needed by the DRAM):

   156.25 ns (five clock (CLK) periods to do the access) −4.5 ns (max delay 74AS00 for CLK~) − 8 ns − 29 ns (ADS~ to RAS~ low max, DP8421A-25 #402) − 7 ns (80286 data setup time #8) − 7 ns (74F245 max delay) = 100.75 ns.

   Therefore the tRAC of the DRAM must be 100.75 ns or less.

6. Determining tCAC during a normal access (CAS~ access time) and column address access time needed by the DRAM:

   156.25 ns (five clock (CLK) periods to do the access) −4.5 ns (max delay 74AS00 for CLK~) − 8 ns (clocked output delay for ADS~ from CLK~) − 82 ns (ADS~ to RAS~ low max, DP8421A-25 #402) − 7 ns (80286 data setup time #8) − 7 ns (74F245 max delay) = 47.75 ns.

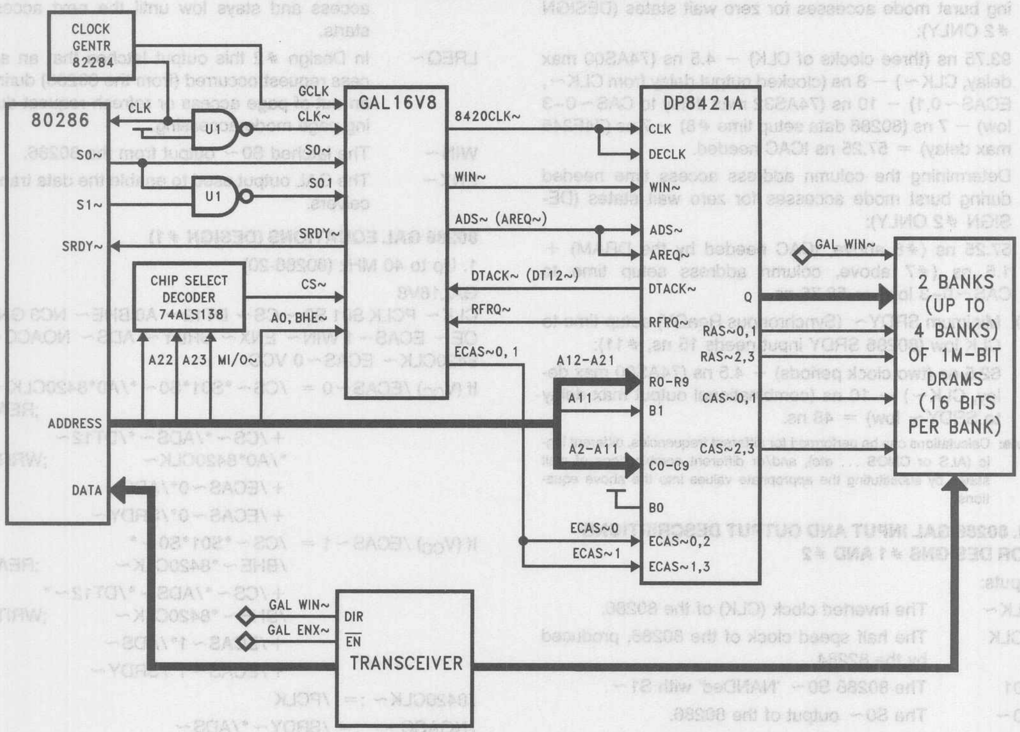
   Therefore the tCAC and the column address access time of the DRAM must be 47.75 ns or less.

7. Determining the column address setup time to CAS~0–3 low (0 ns needed by the DRAMs) during burst mode accesses for zero wait states (DESIGN #2 ONLY):

   31.25 ns (phase 1 in Ts) + 1.25 ns (#2 above, address valid with respect to CLK~ beginning of phase 1 in Ts) + 2 ns (minimum 'D' speed GAL clocked output delay from CLK~, ECAS~0,1) + 2 ns (74AS32 min delay to CAS~0–3 low) = 36.5 ns.

   This gives 1.5 ns column address setup time to CAS~0–3 low (36.5 ns − 35 ns 8421A-25 column address input to output valid, #26).

Interfacing the 80286 to the 8421A



TL/F/10442-1

8. Determining the tCAC (CAS~ access time) needed during burst mode accesses for zero wait states (DESIGN #2 ONLY):

   93.75 ns (three clocks of CLK) − 4.5 ns (74AS00 max delay, CLK~) − 8 ns (clocked output delay from CLK~, ECAS~0,1) − 10 ns (74AS32 max delay to CAS~0–3 low) − 7 ns (80286 data setup time #8) − 7 ns (74F245 max delay) = 57.25 ns tCAC needed.

9. Determining the column address access time needed during burst mode accesses for zero wait states (DESIGN #2 ONLY):

   57.25 ns (#8 above, tCAC needed by the DRAM) + 1.5 ns (#7 above, column address setup time to CAS~0–3 low) = 58.75 ns.

10. Minimum SRDY~ (Synchronous ReaDY) setup time to CLK low (80286 SRDY input needs 15 ns, #11):

    62.5 ns (two clock periods) − 4.5 ns (74AS00 max delay, CLK~) − 10 ns (combinational output max delay to SRDY~ low) = 48 ns.

Note: Calculations can be performed for different frequencies, different logic (ALS or CMOS . . . etc), and/or different combinations of wait states by substituting the appropriate values into the above equations.

VI. 80286 GAL INPUT AND OUTPUT DESCRIPTIONS FOR DESIGNS #1 AND #2

Inputs:

CLK~ The inverted clock (CLK) of the 80286.

PCLK The half speed clock of the 80286, produced by the 82284.

S01 The 80286 S0~ 'NANDed' with S1~.

S0~ The S0~ output of the 80286.

WIN~ The 80286 S0~ input low latched throughout the access cycle.

CS~ The DRAM chip select generated from the 80286 addresses.

DT12~ The DTACK~ output of the 8421A.

A0 The least significant address bit (low byte enable) from the 80286.

BHE~ The high byte enable from the 80286.

RFRQ~ The refresh request output from the 8421A.

HSA~ The High Speed Access output (comparison equal) from the 74ALS6311.

OE~ Output enable of the GAL®.

Outputs:

ECAS~0 The low byte CAS~ enable, this output also toggles during page mode accesses in design #2.

ECAS~1 The high byte CAS~ enable, this output also toggles during page mode accesses in design #2.

SRDY~ This is the ready input to the 80286, it is used to insert wait states into 80286 access cycles.

8420CLK~ This is the CLOCK and DELCLK input to the 8421A. This clock runs at half of the 80286 CLK frequency.

ADS~ This is the ADS~ and AREQ~ inputs to the 8421A. In design #2 this input stays low thru multiple accesses as long as the accesses are within a page.

NOACC~ This GAL output is low at the end of an 80286 access and stays low until the next access starts.

LREQ~ In Design #2 this output latches that an access request occurred (from the 80286) during an out-of-page access or refresh request during page mode accessing.

WIN~ The latched S0~ output from the 80286.

ENX~ The GAL output used to enable the data transceivers.

80286 GAL EQUATIONS (DESIGN #1)

1. Up to 40 MHz (80286-20)

```
GAL16V8
CLK~ PCLK S01 S0~ CS~ DT12~ A0 BHE~ NC3 GND
OE~ ECAS~1 WIN~ ENX~ SRDY~ ADS~ NOACC~
8420CLK~ ECAS~0 VCC
If (VCC) /ECAS~0 =  /CS~*S01*S0~*/A0*8420CLK~
                                                ;READ
                    +/CS~*/ADS~*/DT12~
                    */A0*8420CLK~               ;WRITE
                    +/ECAS~0*/ADS~
                    +/ECAS~0*/SRDY~
If (VCC) /ECAS~1 =  /CS~*S01*S0~*
                    /BHE~*8420CLK~              ;READ
                    +/CS~*/ADS~*/DT12~*
                    /BHE~*8420CLK~              ;WRITE
                    +/ECAS~1*/ADS~
                    +/ECAS~1*/SRDY~
/8420CLK~ :=  /PCLK
/NOACC~    := /SRDY~*/ADS~
              +/NOACC~*/PCLK
              +/NOACC~*CS~*/ADS~
              +/NOACC~*/S01
/ADS~      := /CS~*S01*PCLK
              +/ADS~*SRDY~
/SRDY~     := /CS~*/ADS~*/DT12~*NOACC~*
              /PCLK
              +/SRDY~*/ADS~*NOACC~
/ENX~      := /CS~*/ADS~
/WIN~      := /S0~*S01
              +/WIN~*NOACC~
              +/WIN~*/PCLK
```

80286 PAGE MODE GAL EQUATIONS (DESIGN #2)

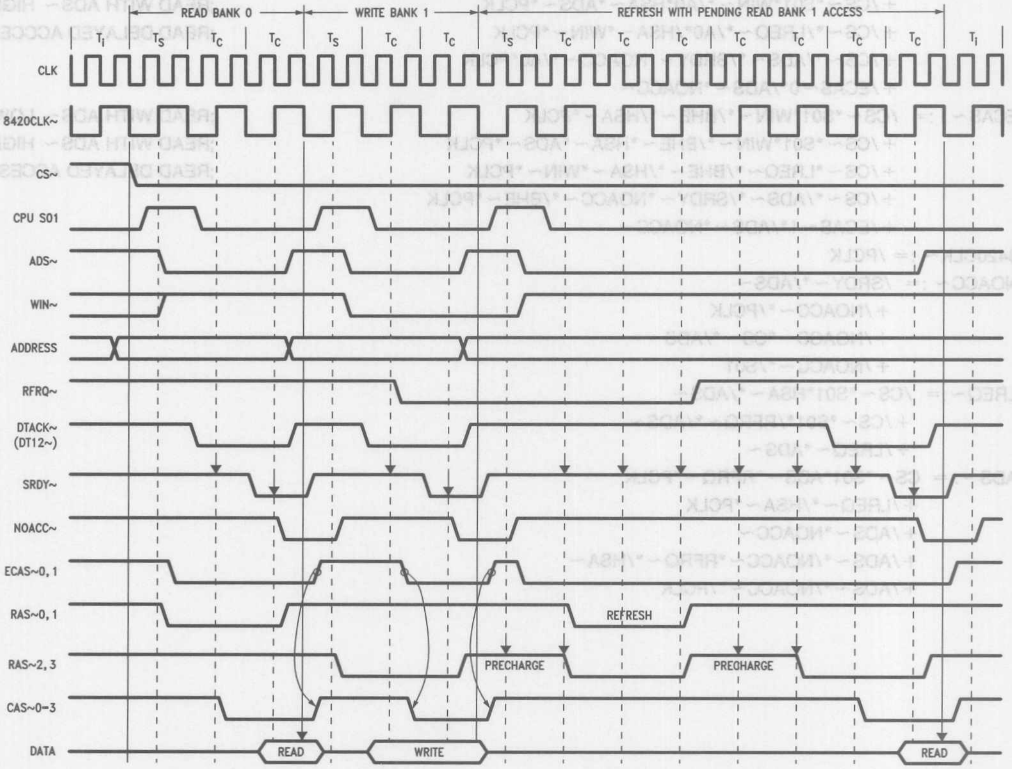
2. Up to 40 MHz (80286-20)

```
GAL16V8
CLK~ PCLK S01 WIN~ CS~ DT12~ RFRQ~ HSA~
A0 GND
OE~ BHE~ ADS~ LREQ~ NOACC~ 8420CLK~
ECAS~1 ECAS~0 SRDY~ VCC
If (VCC)/SRDY~ =  /CS~*/ADS~*
                  /DT12~*NOACC~*8420CLK~
                  +/SRDY~*/ADS~*NOACC~
                  +/SRDY~*/ADS~*8420CLK~
```

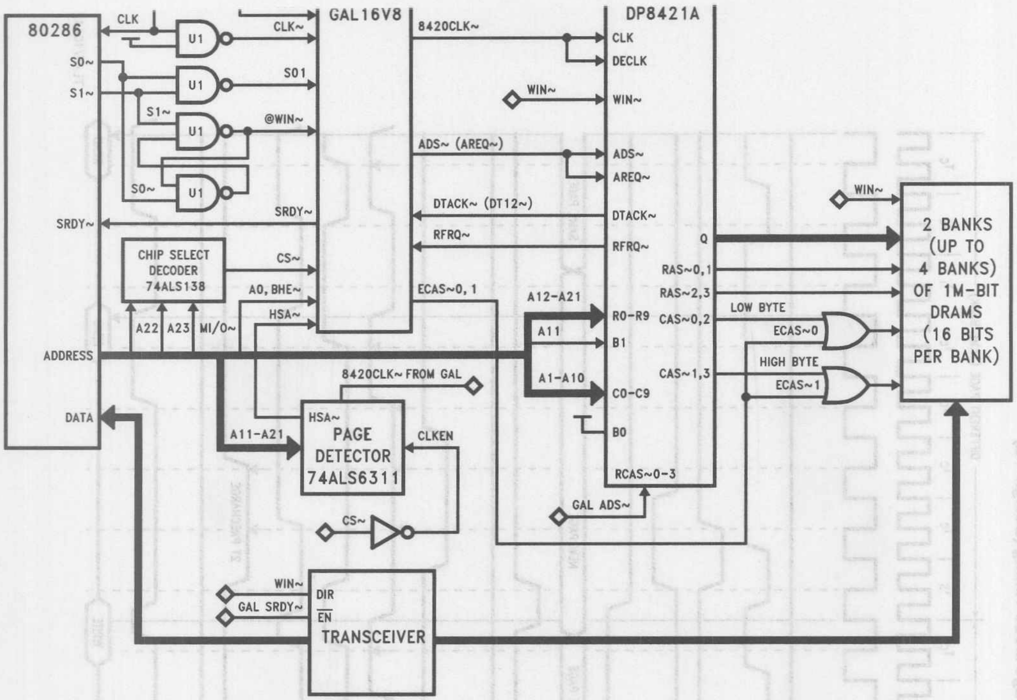
```
/ECAS~0:=  /CS~*S01*WIN~*/A0*/HSA~*PCLK                      ;READ WITH ADS~ LOW
           +/CS~*S01*WIN~*/A0*HSA~*ADS~*PCLK                  ;READ WITH ADS~ HIGH
           +/CS~*/LREQ~*/A0*/HSA~*WIN~*PCLK                   ;READ DELAYED ACCCESS
           +/CS~*/ADS~*/SRDY~*NOACC~*/A0*PCLK
           +/ECAS~0*/ADS~*NOACC~
/ECAS~1:=  /CS~*S01*WIN~*/BHE~*/HSA~*PCLK                     ;READ WITH ADS~ LOW
           +/CS~*S01*WIN~*/BHE~*HSA~*ADS~*PCLK                ;READ WITH ADS~ HIGH
           +/CS~*LREQ~*/BHE~*/HSA~*WIN~*PCLK                  ;READ DELAYED ACCESS
           +/CS~*/ADS~*/SRDY~*NOACC~*/BHE~*PCLK
           +/ECAS~1*/ADS~*NOACC~
/8420CLK~ := /PCLK
/NOACC~ :=  /SRDY~*/ADS~
            +/NOACC~*/PCLK
            +/NOACC~*CS~*/ADS~
            +/NOACC~*/S01
/LREQ~:=  /CS~*S01*HSA~*/ADS~
          +/CS~*S01*/RFRQ~*/ADS~
          +/LREQ~*ADS~
/ADS~:=  CS~*S01*ADS~*RFRQ~*PCLK
         +/LREQ~*/HSA~*PCLK
         +/ADS~*NOACC~
         +/ADS~*/NOACC~*RFRQ~*/HSA~
         +/ADS~*/NOACC~*/PCLK
```

2

80286/DP8421A Page Mode Timing (Design #1)



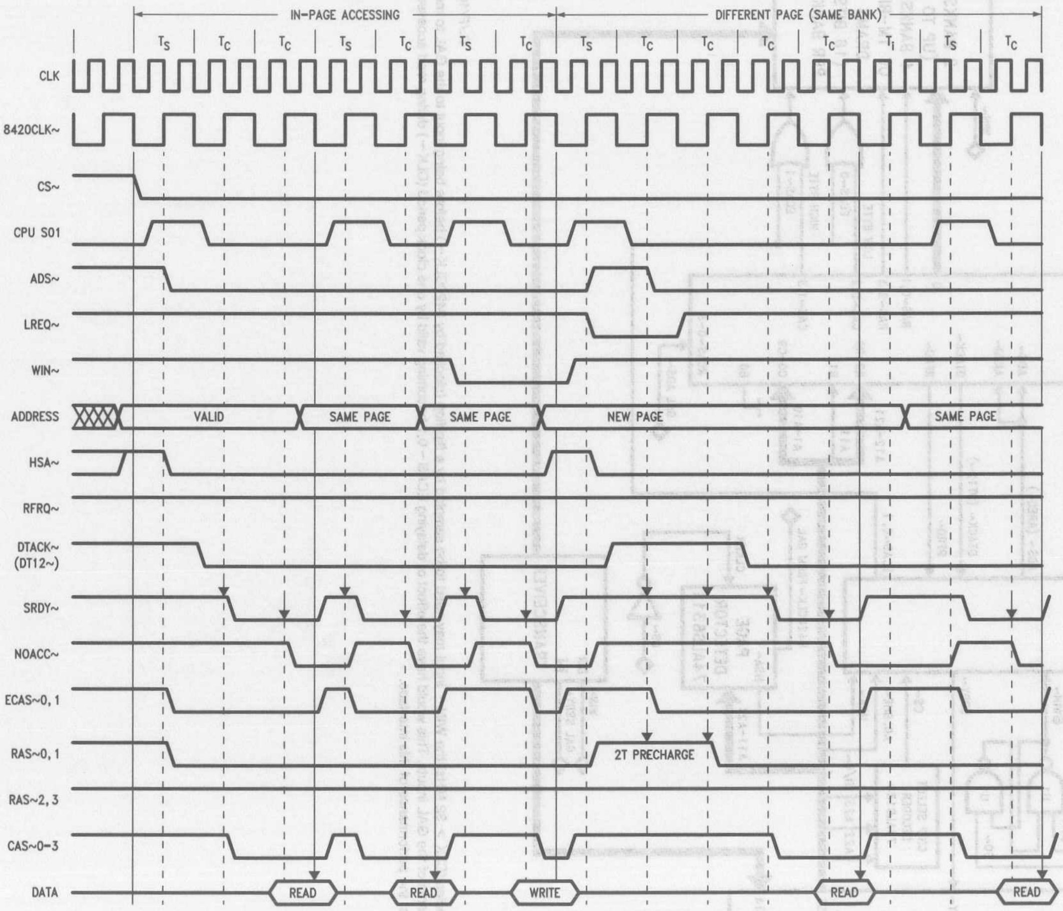
TL/F/10442-2



TL/F/10442-3

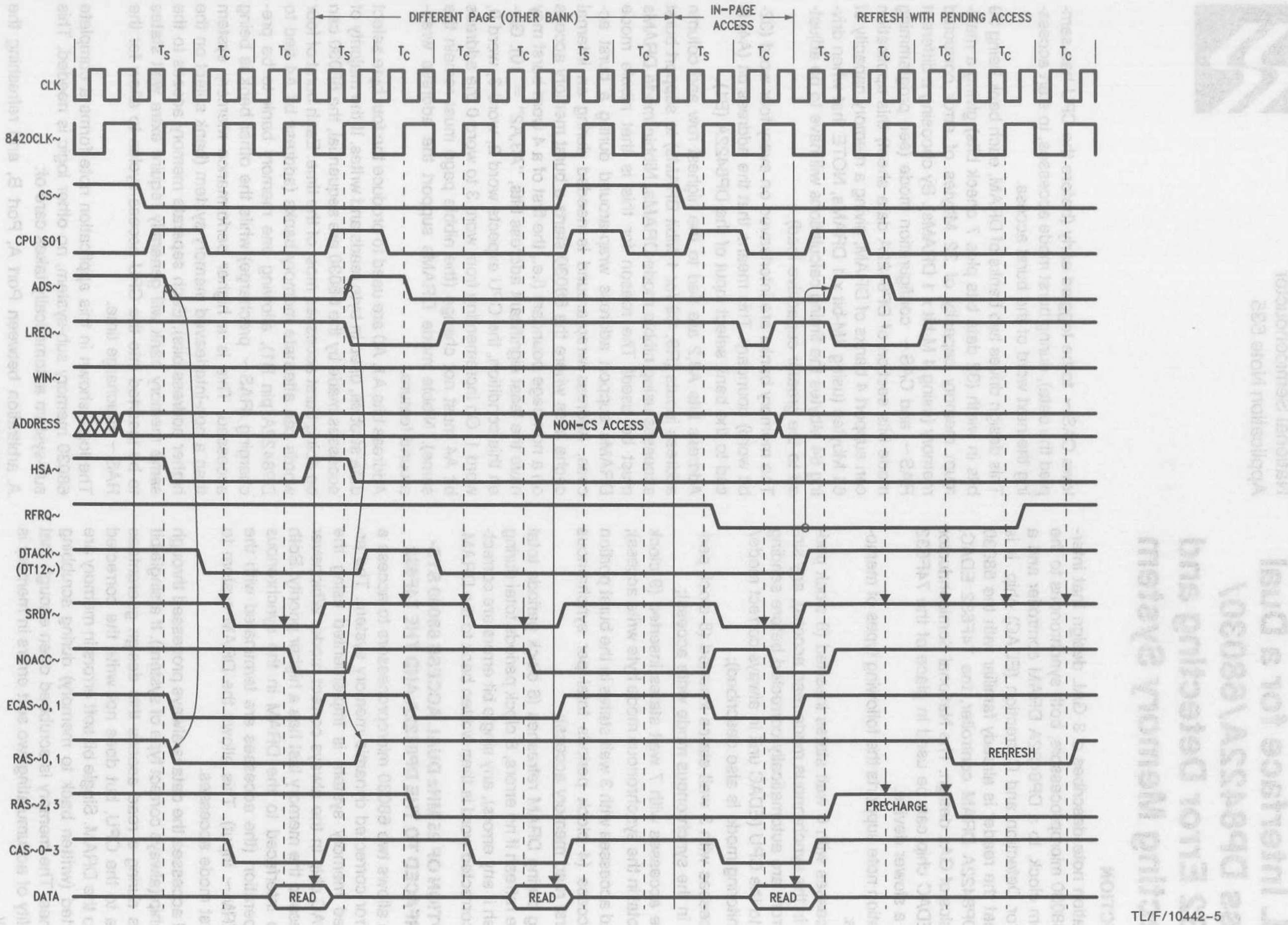
@At high frequencies (CLK > 32 MHz) the WIN~ input may need to be sampled by a flip-flop (clocked by 8420CLK~) before being input to the GAL to meet the setup requirements of the GAL inputs. This would have the effect of delaying ECAS~0,1 becoming valid by one clock period (CLK~) during read accesses, this would not affect the performance of this interface.

80286/DP8421A Page Mode Timing (Design #2)



TL/F/10442–4

80286/DP8421A Page Mode Timing (Design #2)



TL/F/10442-5

A GAL Interface for a Dual Access DP8422A/68030/74F632 Error Detecting and Correcting Memory System

National Semiconductor
Application Note 535

I INTRODUCTION

This application note describes a 3 GAL design that interfaces two 68030 microprocessors, both synchronous to the same system clock, to a DP8422A DRAM controller and a 74F632 Error Detection and Correction (EDAC) chip. It is assumed that the reader is already familiar with the 68030 CPU, the DP8422A DRAM controller, the 74F632 EDAC, and the basics of GAL design. The National Semiconductor DP8402A EDAC chip can be used in place of the 74F632 though it is a slower device.

This application note supports the following types of memory accesses:

1. Read accesses with 6 wait states inserted (8 clock periods total in the synchronous mode read access), any single bit errors are automatically corrected before sending the data to the CPU (EDAC unit in always correct mode/error monitoring mode is also described);
2. Write accesses with 3 wait states inserted (5 clock periods total in the synchronous mode write access);
3. Byte write accesses with 7 wait states inserted (9 clock periods total in the synchronous mode byte write access);
4. Burst read accesses with 3 wait states in the burst portion of the access (4 clock periods total per synchronous mode burst read memory access);
5. Scrubbing during DRAM refreshes (6 clock periods total during the refresh if no errors, 8 clock periods total during the refresh if any errors), any single bit errors are corrected. The corrected word is then written back to the DRAM.

II DESCRIPTION OF 25 MHz DUAL ACCESS 68030 SYSTEM INTERFACED TO THE DP8422A AND THE 74F632

This design allows two 68030 microprocessors to access a common error corrected dynamic memory system. The error corrected memory system is implemented using the 74F632 EDAC chip in the always correct mode. Whichever 68030 accessed the memory last has a higher priority. Both 68030s are interfaced to the DRAM in the synchronous mode of operation (the accesses are terminated with the 68030 STERM~ input). This allows the DRAM system to support burst mode accesses.

During read accesses the data is always processed through the EDAC chip (always correct type of system). If a single bit error occurs during a read access this design guarantees correct data to the CPU, but does not write the corrected data back to the DRAM. Single bit soft errors in memory are only corrected (written back to memory) during scrubbing type refreshes. The memory is scrubbed often enough that the probability of accumulating two soft errors in memory is very unlikely.

During read accesses the data is always processed through the 74F632 EDAC chip (i.e., the EDAC data buffers are enabled to provide the data to the CPU). The 74F632 is always put into latch and correct mode during read accesses, even though the data from the memory may be correct. This allows CAS~ to be toggled early (before the CPU has sampled the data), during burst mode accesses, to start accessing the next word of the burst access.

This design drives two banks of DRAM, each bank being 39 bits in width (32 data bits plus 7 check bits) giving a maximum memory capacity of 32 Mbytes of error corrected memory (using 4 M-bit x 1 DRAMs). By choosing a different RAS~ and CAS~ configuration mode (see programming mode bits section of DP8422A data sheet) this application can support 4 banks of DRAM, giving a memory capacity of 64 Mbytes (using 4M-bit x 1 DRAMs, NOTE that when driving 64 Mbytes the timing calculations will have to be adjusted to the greater capacitive load).

The memory banks are interleaved on every four word (32-bit word) boundary. This means that the address bit (A4) is tied to the bank select input of the DP8422A (B1).

Address bits A3,2 are tied to the highest row and column address inputs (R9, C9 for 1 Mbit DRAMs) to support burst accesses using nibble mode DRAMs. Nibble mode DRAMs must be used! The reason for this is that nibble mode DRAMs support address wrap-around during a burst access. Address wrap-around is needed during an internal cache miss where the 68030 starts a burst memory access on a non-page boundary (i.e., the first of a 4 word burst may have the least significant address bits, "A3,A2" = 10). Given this condition, the CPU expects word 2, word 3, word 0, word 1. On incrementing from word 3 to word 0 the address bit A4 must not change (the nibble page must remain the same). Nibble mode DRAMs support the address wrap-around feature.

Address bits A1, A0 are used to produce the four byte select data strobes, used in byte reads and writes. If the majority of accesses made by the 68030 are sequential, the 68030 can be doing burst accesses most of the time. Each burst of four words can alternate memory banks (address bit A4 tied to DP8422A pin B1), allowing one memory bank to be precharging (RAS~ precharge) while the other bank is being accessed. This is a higher performance memory system than a non-interleaved memory system (bank select on the higher address bits). Each separate memory access to the same memory bank will generally require extra wait states to be inserted into the CPU access cycles to allow for the RAS~ precharge time.

The logic shown in this application note forms a complete 68030 memory sub-system, no other logic is needed. This sub-system automatically takes care of:

A. arbitration between Port A, Port B, and refreshing the DRAM;

B. the insertion of wait states to the processor (Port A and Port B) when needed (i.e., if RAS~ precharge is needed, refresh is happening during a memory access, the other Port is currently doing an access . . . etc.);

C. performing byte writes and reads to the 32-bit words in memory;

D. normal and burst access operations.

By making use of the enable input on the 74AS244 buffer, this application allows dual access applications. The addresses and chip select are TRI-STATE® through this buffer, the write input (WIN~), lock input (LOCK~), and ECAS0–3 ~ inputs must also be able to be TRI-STATE (another 74AS244 could be used for this purpose). By multiplexing the above inputs (through the use of the above parts and similar parts for Port B) the DP8422A allows dual accessing to be performed.

III ANOTHER OPTION FOR A 68030 25 MHz DUAL ACCESS EDAC DESIGN: THE EDAC ERROR MONITORING METHOD IN CONJUNCTION WITH THE 68030 ASYNCHRONOUS LATE RETRY FEATURE

The 68030 dual access EDAC system design could use the error monitoring method in conjunction with the 68030 asynchronous late retry feature, instead of the always correct method (design shown in this application note). The error monitoring method can yield a slight improvement in system performance.

By using the error monitoring method of error correction single read accesses or the first read access during a burst access can be shortened by one clock period, allowing a synchronous read access to have only 5 wait states inserted, 7 clock periods total (compared to 6 wait states, 8 clock periods total when doing the always correct method). All other types of accesses (burst reads, byte writes, word writes, refresh scrubbing) will execute in the same number of clock cycles, and in the same manner as described in this application note.

Read accesses can save one wait state because the data from the DRAM memory is assumed to be correct in the error monitoring system design. Therefore the DRAM data is given directly to the CPU instead of running it through the EDAC chip as was done in the always correct method.

In order to do this design it is required that the asynchronous late retry feature of the 68030 and registered transceivers (74F646) be employed.

The asynchronous late retry feature of the 68030 involves pulling the 68030 input signals "BERR~ and HALT~" both low before the falling clock edge of the last clock cycle of the access. Given that this is done the 68030 will suspend all bus activity until HALT~ is brought high and then will retry the aborted bus cycle (unless that access is not currently needed by the CPU). This feature is useful for the case where an error is detected in the DRAM data. In this case BERR~ and HALT~ are brought low until the data from the DRAM is corrected (by the EDAC chip) and written back to the DRAM. BERR~ and HALT~ are then brough high to continue CPU processing.

Registered transceivers (74F646) are necessary during burst mode read accesses because CAS~ transitions high before the CPU has sampled the DRAM data. The registered transceivers hold the data valid until the CPU samples it during these cases.

A read, read with a single bit error, and burst read access timing are shown at the end of this application note implementing the error monitoring method. The user can see how these access cycles differ from the always correct method access cycles.

IV 68030 25 MHz DUAL ACCESS DESIGN, PROGRAMMING MODE BITS

Programming Bits	Description
R0 = 1	RAS~ low four clocks, RAS~ precharge of three clocks
R1 = 1	
R2 = 1	DTACK~ 1 is chosen. DTACK~ low first rising CLK edge after access RAS~ is low.
R3 = 0	
R4 = 0	No WAIT states during burst accesses
R5 = 0	
R6 = 0	If WAITIN~ = 0, add one clock to DTACK~. WAITIN~ may be tied high or low in this application depending upon the number of wait states the user desires to insert into the access.
R7 = 1	Select DTACK~
R8 = 1	Non-interleaved mode
R9 = X	
C0 = X	Select based upon the input "DELCLK" frequency. Example: if the input clock frequency is 20 MHz then choose C0,1,2 = 0,0,0 (divide by ten, this will give a frequency of 2 MHz). If DELCLK of the DP8422A is over 20 MHz do an initial divide by two externally and then run that output into the DELCLK input and choose the correct divider.
C1 = X	
C2 = X	
C3 = X	
C4 = 0	RAS~ groups selected by "B1". This mode allows two RAS~ outputs to go low during an access, and allows byte writing in 32-bit words.
C5 = 0	
C6 = 1	
C7 = 1	Column address setup time of 0 ns
C8 = 1	Row address hold time of 15 ns
C9 = 1	Delay CAS~ during write accesses to one clock after RAS~ transitions low
B0 = 1	Fall-thru latches
B1 = 1	Access mode 1
ECAS0~ = 0	Non-extend CAS~

0 = Program with low voltage level

1 = Program with high voltage level

X = Program with either high or low voltage level (don't care condition)

V 68030 25 MHz WORST CASE TIMING CALCULATIONS

The worst case access is an access from Port B. This occurs because the time to RAS~ and CAS~ low is longer for the Port B access than; a Port A access, a refresh with scrubbing access, or an access which has been delayed from starting (due to refresh, RAS~ precharge time, or the other Port accessing memory).

A. Worst case time to RAS~ low from the beginning of an access cycle:

40 ns (T1 clock period of 68030) + 10 ns (maximum combinational output delay to produce AREQB~) + 41 ns (DP8422A-25 parameter #102, AREQ~ to RAS~ delay maximum) = 91 ns

B. Worst case time to CAS~ low from the beginning of an access cycle:

40 ns + 10 ns + 94 ns (DP8422A-25 parameter #118a, AREQB~ to CAS~ delay maximum) = 144 ns

C. Worst case time to DRAM data valid:

144 ns (from "B" above, maximum time to CAS~) + 50 ns (CAS~ access time "t_{CAC}" for a typical 100 ns DRAM) = 194 ns

D. Worst case time to data valid on the EDAC data bus:

194 ns (from "C" above) + 7 ns (74AS244 maximum delay) = 201 ns

E. Worst case time until the error flags are valid from the 74F632:

201 ns (from "D" above) + 31 ns (74F632 maximum time to error flags valid) = 232 ns

F. Worst case time until corrected data is valid from the 74F632:

201 ns (from "D" above) + 28 ns (74F632 maximum time from data in to corrected data out) = 229 ns

G. Worst case time until corrected data is available at the CPU:

229 ns (from "F" above) + 7 ns (74F245 maximum delay) = 236 ns

VI 68030 25 MHz DUAL ACCESS DESIGN, TIMING CALCULATIONS

1. Minimum ADS~ low setup time to CLOCK~ high for DTACK~ logic to work correctly (DP8422A-25 needs 25 ns, parameter #400b):

40 ns (one clock period) − 10 ns (combinational output maximum that produces AREQ~, ADS~) = 30 ns

2a. Minimum address setup time to ADS~ low (DP8422A-25 needs 14 ns, parameter #404):

40 ns (one clock period) − 20 ns (assumed 68030 max time to address valid from CLK high) − 6.2 ns (74AS244 buffer delay max) + 2.5 ns (minimum combinational output delay that produces AREQ~, ADS~) = 16.3 ns

2b. Minimum address setup time to CLK high (used in #3B calculation below):

40 ns (one clock period) − 20 ns (assumed 68030 max time to address valid from CLK high) − 6.2 ns (74AS244 buffer delay max) = 13.8 ns

3a. Minimum CS~ setup time to ADS~ low (DP8422A-25 needs 5 ns, parameter #401):

16.3 ns (#2a) − 9 ns (max 74AS138 decoder) = 7.3 ns

3b. Minimum CS~ setup time to CLK high (GAL equations need 0 ns):

13.8 ns (#2b) − 9 ns (max 74AS138 decoder) = 4.8 ns

4. Determining t_{RAC} during a normal access (RAS~ access time needed by the DRAM):

200 ns (five and one-half clock periods to get data from the DRAM to the 74F632 data inputs) − 3 ns (74F632 data setup time to mode input S0 high) + 2.5 ns (minimum combinational output delay for "S0") − 84 ns (from "A" of worst case times, from the beginning of the access to RAS~ low) − 6.2 ns (74F244 DRAM buffer delay maximum) = 129.3 ns

Therefore the t_{RAC} of the DRAM must be 129.3 ns or less.

5. Determining t_{CAC} during a normal access (CAS~ access time) and column address access time needed by the DRAM:

220 ns (five and one-half clock periods to get data from the DRAM to the 74F632 data inputs) − 3 ns (74F632 data setup time to mode input S0 high) + 2.5 ns (minimum combinational output delay for "S0") − 138 ns (from "B" of worst case times, from the beginning of the access to CAS~ low) − 6.2 ns (74F244 DRAM buffer delay maximum) = 75.3 ns

Therefore the t_{CAC} of the DRAM must be 75.3 ns or less.

6. Determining the nibble mode access time needed during a burst access:

100 ns (two and one-half clock periods to do the burst) − 8 ns (clocked output delay maximum for ENCAS~ output) − 27 ns (DP8422A-25 ECASn~ to CASn~ asserted maximum, parameter #14) − 3 ns (74F632 data setup time to mode input S0 high) + 2.5 ns (minimum combinational output delay for "S0") − 6.2 ns (74F244 DRAM buffer delay maximum) = 58.3 ns

Therefore the nibble mode access time of the DRAM must be 58.3 ns or less

7. Maximum time to DTACK1~ low (GAL16V8 needs 10 ns setup to CLK):

40 ns (One clock) − 28 ns (DTACK2~ low from CLK high on DP8422A-25, parameter #18) = 12 ns

8. Minimum STERM~ setup time to CLK (0 ns to CLK rising edge is needed by the 68030):

20 ns (one-half clock period) − 10 ns (combinational output maximum) = 10 ns

**Note: That calculations can be performed for different frequencies and/or different combinations of wait states by substituting the appropriate values into the above equations.

VII GAL INPUT DESCRIPTIONS

BCLK System Clock

CLK System Clock

CSA~ Chip Select from Port A 68030

ASA~ Address Strobe from Port A 68030

CSASA~ Chip selected access request from Port A 68030

CSB~ Chip Select from Port B 68030

ASB~ Address Strobe from Port B 68030

DTACK~ Data Transfer ACKnowledge for Port B accesses

ATACKB~ Transfer ACKnowledge for Port B accesses

R Read/Write~ (R/W~) indicator from the currently granted CPU

CBREQ~ Cache Burst REQuest indicator from the currently granted CPU

WCBREQ~ When low this signal indicates either a write access or a non-burst access

RFIP~ Indicates that a DRAM refresh is in progress

RAS0~ RAS0~ output from the DP8422A DRAM controller

WORD~ Indicates a word access (32 bits) as opposed to a byte or multi-byte access (less than 32 bits)

GRANTB GRANTB output from the DP8422A DRAM controller, when high this output indicates that Port B currently is granted to access the DRAM

VIII GAL OUTPUT DESCRIPTIONS

AREQ~ DRAM Access REQuest for Port A 68030

AREQB~ DRAM Access REQuest for Port A 68030

COUNT~ The enable for the shift register counter (outputs D1-6~)

D1-6~ Shift register counter, these outputs are used to drive the GAL control outputs in the proper sequence for each access (Port A, Port B, refresh) and are clocked outputs

ENCAS~ This output, when low, enables the CAS~ outputs of the DP8422A DRAM controller and is a clocked output

EXRF~ This output is used to EXtend the ReFresh cycle to allow an access from one of the banks of DRAM, if an error occurs (ERR~ low) the refresh cycle is extended even longer to allow the corrected data to be written back to memory

S0~ This output controls the S0 mode input of the 74F632

S1~ This output controls the S1 mode input of the 74F632

TRAN_EN~ This output is used to enable the data transceivers for the currently enabled Port (A or B)

OEB~ This output is used to drive the OEB0-3~ inputs of the 74F632 to provide byte output control of the latched corrected data

OECB~ This output controls when to enable the check bits out of the 74F632

LEDB0~ This output is used to latch the corrected data in the output latches of the 74F632

STERMA~ This output is used to insert synchronous wait states to the Port A 68030

STERMB~ This output is used to insert synchronous wait states to the Port B 68030

SERR~ This output latches the fact that the 74F632 detected an error in the data it read from the DRAM

BERR~ This output latches that the 74F632 detected a multiple bit error in the data it read from the DRAM

WE~ This output controls write enable to the DRAMs

IX 68030 25 MHz DUAL ACCESS EDAC SYSTEM DESIGN GAL EQUATIONS IN ABEL FORMAT

DP1 device "GAL16V8"

```
BCLK       pin 1;    VCC       pin 20;
CLK        pin 2;    AREQ~     pin 19;
CSASA~     pin 3;    AREQB~    pin 18;
CSB~       pin 4;    D1~       pin 17;
ASB~       pin 5;    D2~       pin 16;
DTACK~     pin 6;    D3~       pin 15;
ATACKB~    pin 7;    ENCAS~    pin 14;
WCBREQ~    pin 8;    COUNT~    pin 13;
RFIP~      pin 9;    RAS0~     pin 12;
GND        pin 10;   OE~       pin 11;
```

EQUATIONS

```
!AREQ~ = !CSASA~ & CLK
     #!AREQ~ & !CSASA~
     #!AREQ~ & !CLK;

!AREQB~ = !CSB~ & !ASB~ & CLK
     #!AREQB~ & !CSB~ & !ASB~
     #!AREQB~ & !CLK;

!COUNT~ = !AREQ~ & !DTACK~ & !CSASA~
     #!AREQB~ & !ATACKB~ & !ASB~
     #!RFIP~ & !RAS0~;

!D1~ := !AREQ~ & !DTACK~
      #!ATACKB~ & !AREQB~
      #!RFIP~ & !RAS0;

!D2~ :!D1~ & D3~ & !COUNT~
      #D3~ & !AREQ~ & !DTACK~ & RFIP~;

!D3~ := !D2~ & !COUNT~;

!ENCAS~ := !WCBREQ~
             # D1~
             # !D2~
             # D3~
             # !RFIP~;
```

DP2 device "GAL16L8D"

```
BCLK      pin 1;    VCC        pin 20;
R         pin 2;    EXRF~      pin 19;
WORD~     pin 3;    S0         pin 18;
GRANTB    pin 4;    S1         pin 17;
RFIP~     pin 5;    TRAN_EN~   pin 16;
SERR~     pin 6;    OEB~       pin 15;
D2~       pin 7;    OECB~      pin 14;
D5~       pin 8;    STERMA~    pin 13;
D6~       pin 9;    STERMB~    pin 12;
GND       pin 10;   OE~        pin 11;
```

```
!S0 = !R & !WORD~ & RFIP~
      # !D2~ & D5~
      # !S0 & BCLK
      # !D5~ & !BCLK
      # !S0 & !D5~
      # !S0~ &! D6~
      # !S1 & !SERR~ & !RFIP~;

!S1 = !R & !WORD~ & RFIP~
      # !D5~ & !BCLK
      # !S1 & !D5~
      # !S1 & !D6~ & !R & WORD~
      # !S1 & !D6~ & !RFIP~
      # !S1 & !SERR~ & !RFIP~;

!TRAN__EN~ = R & !D5~ & !BCLK & RFIP~
      # !TRAN__EN~ & R & !D5~ & D6~ & RFIP~
      # R & !D5~ & !STERMA~ & RFIP~
      # R & !D5~ & !STERMB~ & RFIP~
      # !R & !WORD~ & !S1 & RFIP~
      # !R & WORD~ & !D5~ & !BCLK & RFIP~
      # !TRAN__EN~ & !R & WORD~ & !D5~ &
        RFIP~
      # !TRAN__EN~ & !R & WORD~ & !D6~ &
        RFIP~;

!OEB~ = R & !D5~ & !BCLK
      # !OEB~ & R & !D5~
      # !RFIP~ & !D5~ & !BCLK & !SERR~
      # !OEB~ & !RFIP~ & !D5~ & !SERR~
      # !OEB~ & !RFIP~ & !D6~ & !SERR~
      # !R & WORD~ & !D5~ & !BCLK
      # !OEB~ & !R & WORD~ & !D5~
      # !OEB~ & !R & WORD~ & !D6~;

!OECB~ = !R & !WORD~ & RFIP~ & !S1
      # !RFIP~ & !D5~ & !BCLK & !SERR~
      # !OECB~ & !RFIP~ & !D5~ & !SERR~
      # !OECB~ & !RFIP~ & !D6~ & !SERR~
      # !R & WORD~ & !D5~ & !BCLK
      # !OECB~ & !R & WORD~ & !D5~
      # !OECB~ & !R & WORD~ & !D6~;

!STERMA~ = R & RFIP~ & !D5~ & D6~ &
           !GRANTB~ & !BCLK
      # !STERMA~ & R & RFIP~ & !D5~ &
        !GRANTB~ & BCLK
      # !R & !WORD~ & RFIP~ & !D2~ & D6~ &
        !GRANTB~ & !BCLK
      # !STERMA~ & !R & !WORD~ & RFIP~ &
        !D2~ & D6~ !GRANTB & BCLK
      # !R & WORD~ & RFIP~ & !D5~ & !D6~ &
        !GRANTB~ & !BCLK
      # !STERMA~ & !R & WORD~ & RFIP~ & !D6 &
        !GRANTB & BCLK;
```

```
        !GRANTB & !BCLK
      # !STERMB~ & !R & !WORD~ & RFIP~ &
        !D2~ & D6~ GRANTB & BCLK
      # !R & WORD~ & RFIP~ & !D5~ & !D6~ &
        GRANTB~ & !BCLK
      # !STERMB~ & !R & WORD~ & RFIP~ &
        !D6 & GRANTB & BCLK;
```

DP3 device "PAL16R4D"

BCLK	pin 1;	VCC	pin 20;
CLK	pin 2;	LEDB0~	pin 19;
S0~	pin 3;	SERR~	pin 18;
S1~	pin 4;	WE~	pin 17;
ERR~	pin 5;	D4~	pin 16;
MERR~	pin 6;	D5~	pin 15;
COUNT~	pin 7;	D6~	pin 14;
D2~	pin 8;	BERR~	pin 13;
D3~	pin 9;	OECB~	pin 12;
GND	pin 10;	OE~	pin 11;

EQUATIONS

```
!LEDB0 = !D2~ & !S0~ & S1~ & !CLK
      #!LEDB0~ & !D3~ & !S0~
      #!LEDB0~ & !CLK

!SERR~ = !D4~ & S0~ & S1~ & !COUNT~ &
         !ERR~ & CLK
      #!SERR~ & !COUNT~;

!BERR~ = D4~ & S0~ & S1~ & !COUNT~ &
         !MERR~ & CLK
      #!BERR~ & !COUNT~;

!WE~ := !S1~ & !D2~ & D3~ & !COUNT~ & !OECB~;

!D4~ := !D3~ & !COUNT~;

!D5~ := !D4~ & !COUNT~;

!D6~ := !D5~ & !COUNT~;
```

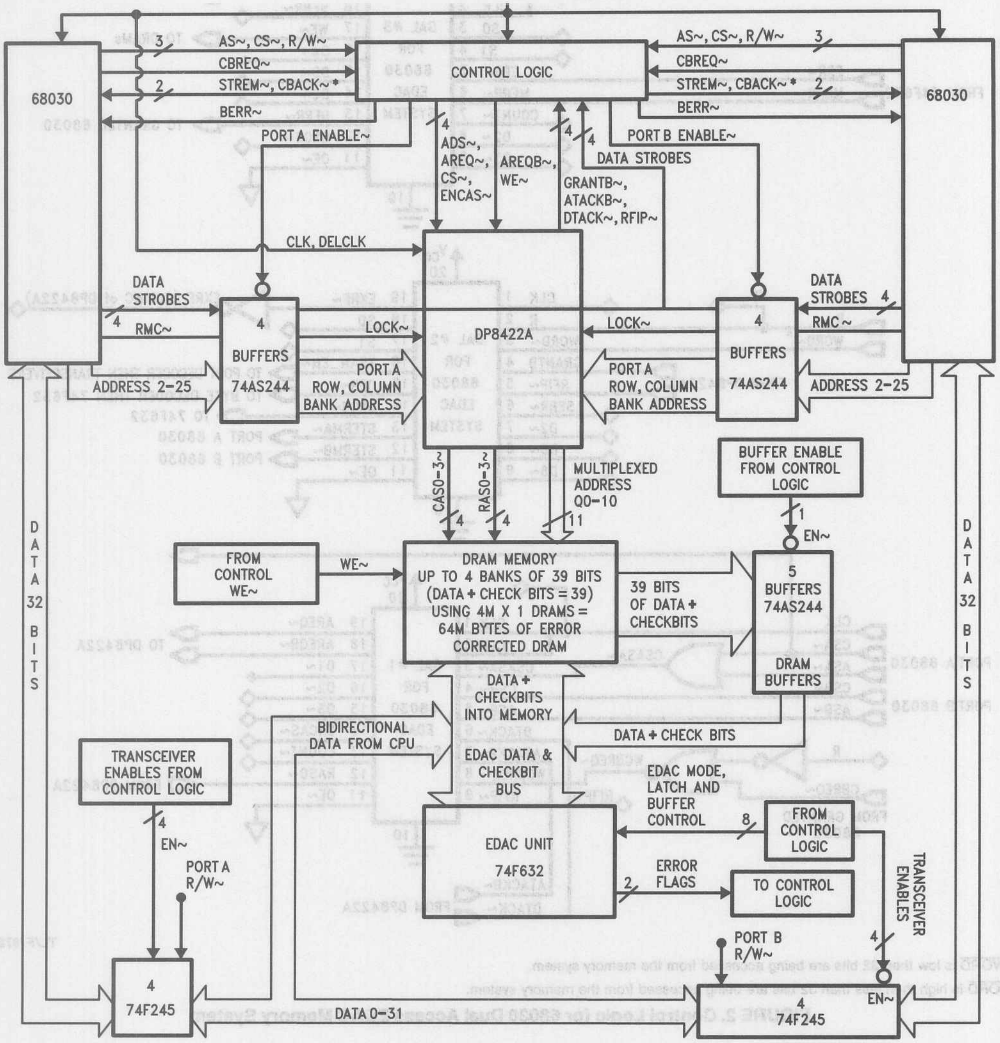
Key: Reading PAL equations

EXAMPLE EQUATIONS:

```
!AREQ~ = !CSASA~ & CLK
         # !AREQ~ & !CSASA~
         # !AREQ~ & !CLK~
```

This example reads: the output "AREQ~" will transition low given that one of the following conditions are valid;

1. the input "CSASA~" is low AND the input "CLK" is high, OR
2. the output "AREQ~" is low AND the input "CSASA~" is low, OR
3. the output "AREQ~" is low AND the input "CLK" is low.

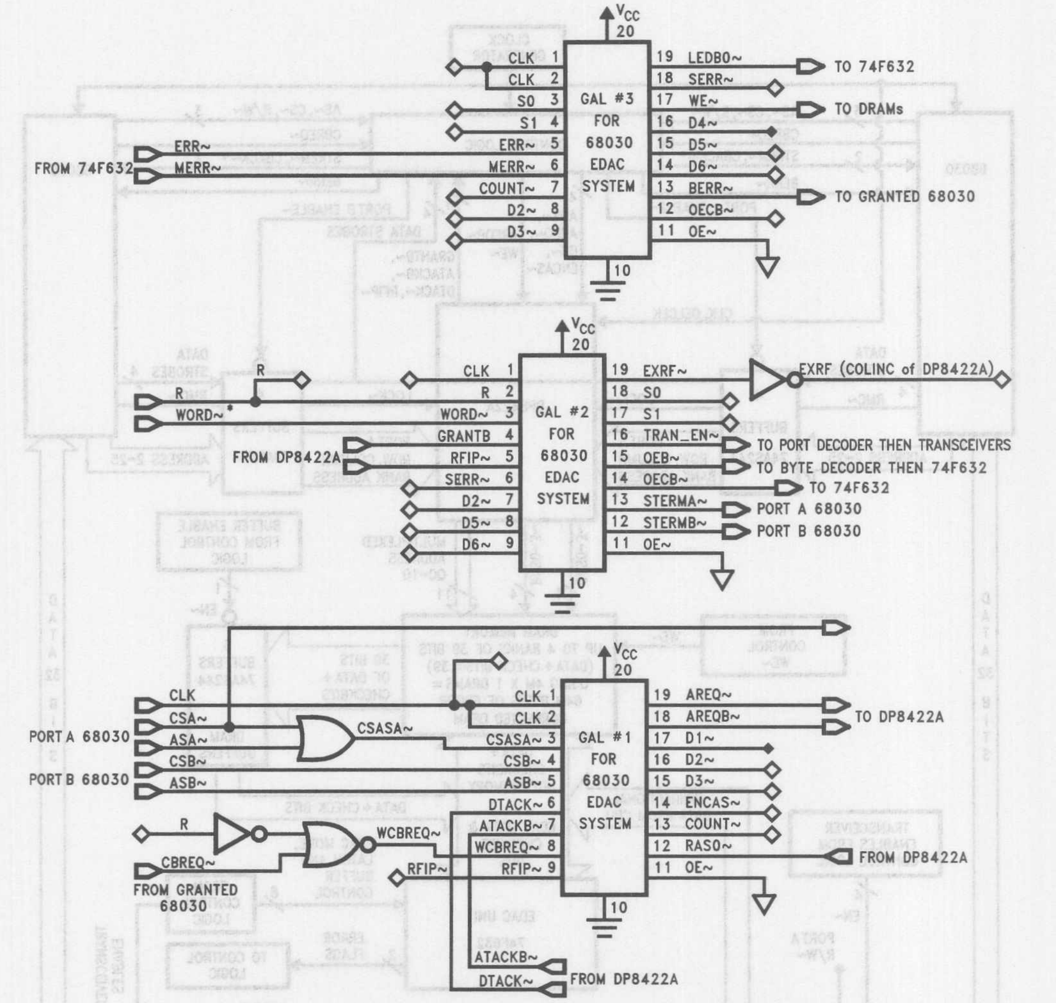


TL/F/9729–1

Control logic in this system needs the following: 3 GAL® devices and some logic gates

*$\overline{\text{CBACK}}$ is tied low back to 68030

FIGURE 1. Block Diagram of Dual Access 68030 Error Detecting and Correcting (74F632) Memory System

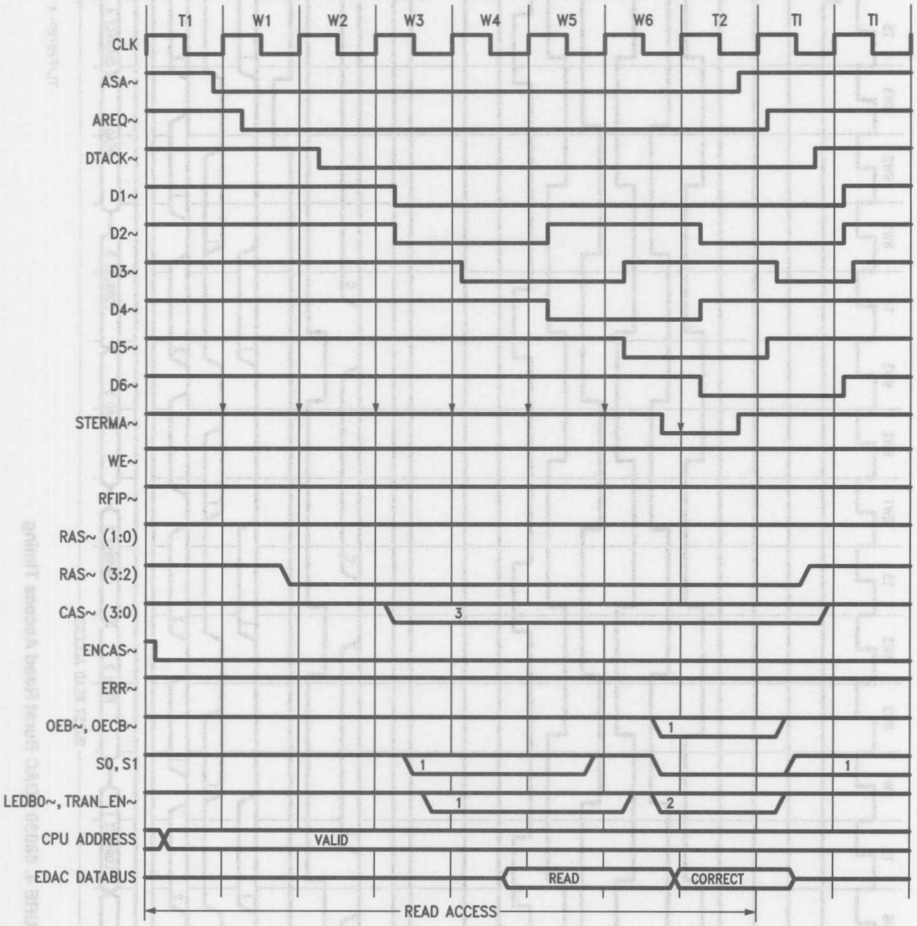


TL/F/9729-2

*If WORD is low then 32 bits are being accessed from the memory system.

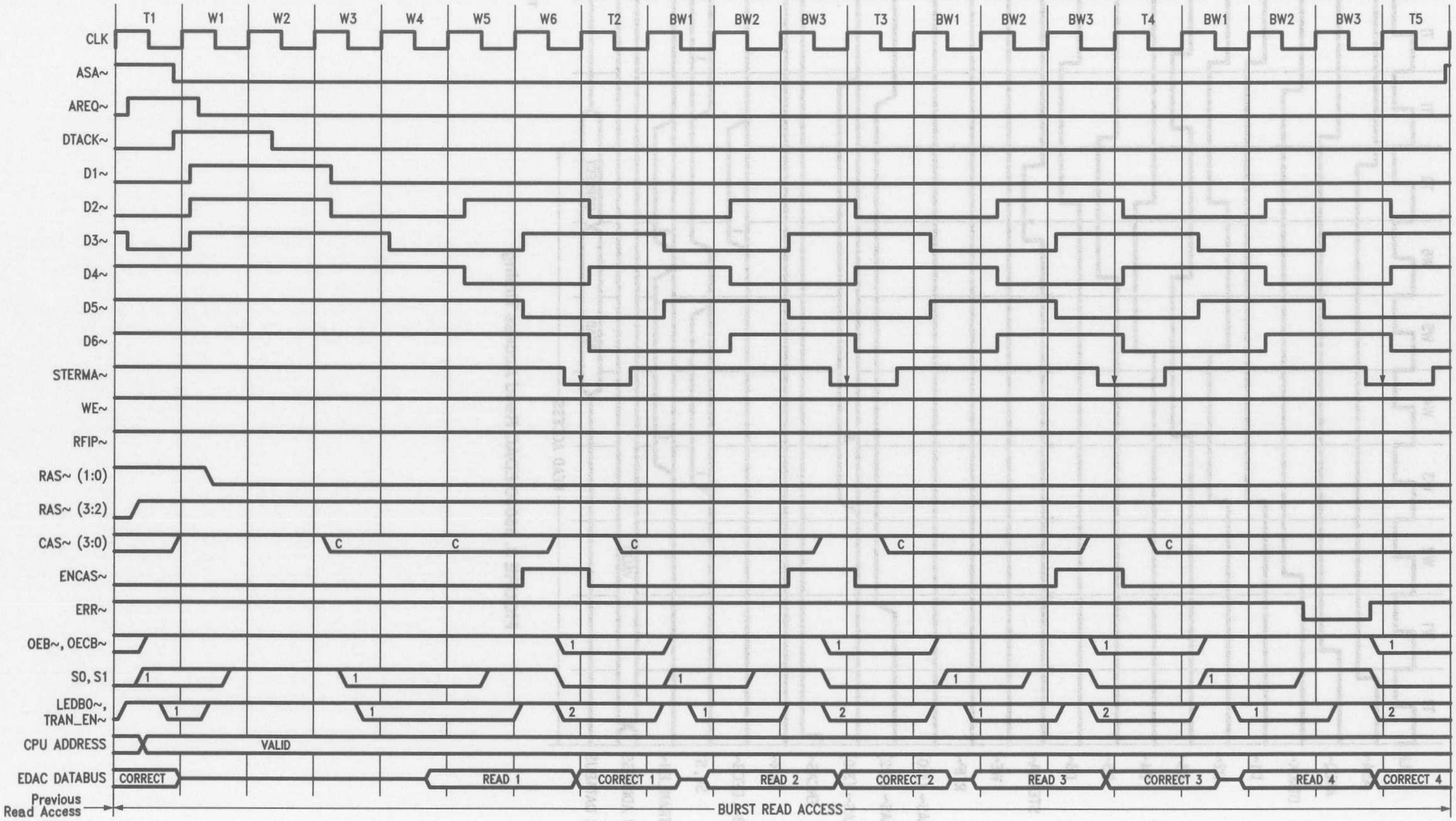
If WORD is high then less than 32 bits are being accessed from the memory system.

FIGURE 2. Control Logic for 68030 Dual Access EDAC Memory System



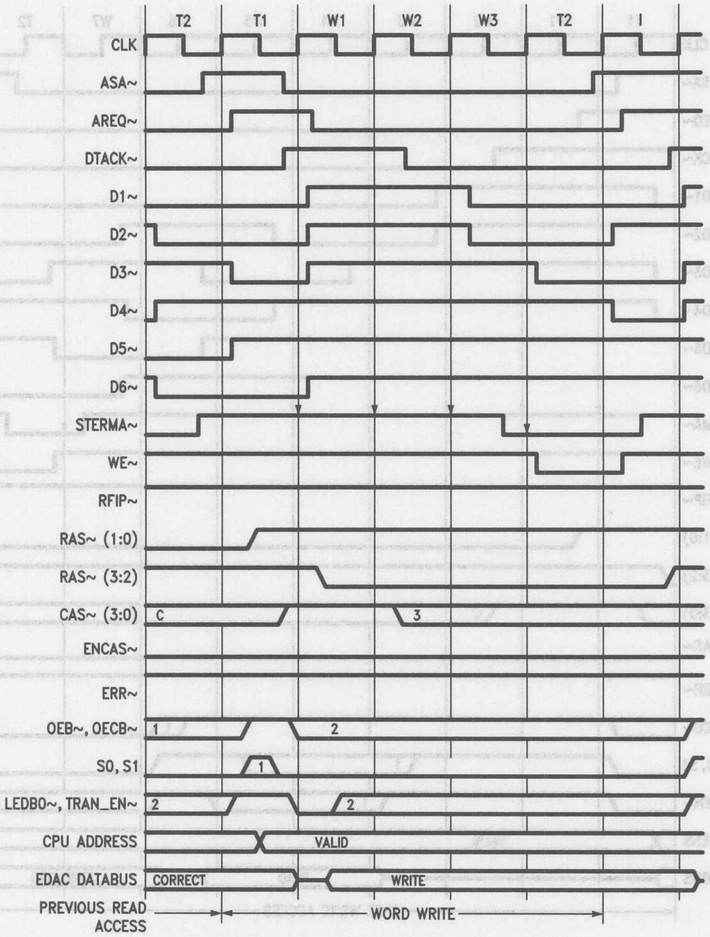
TL/F/9729-3

FIGURE 3. 68030 EDAC Read Access Timing



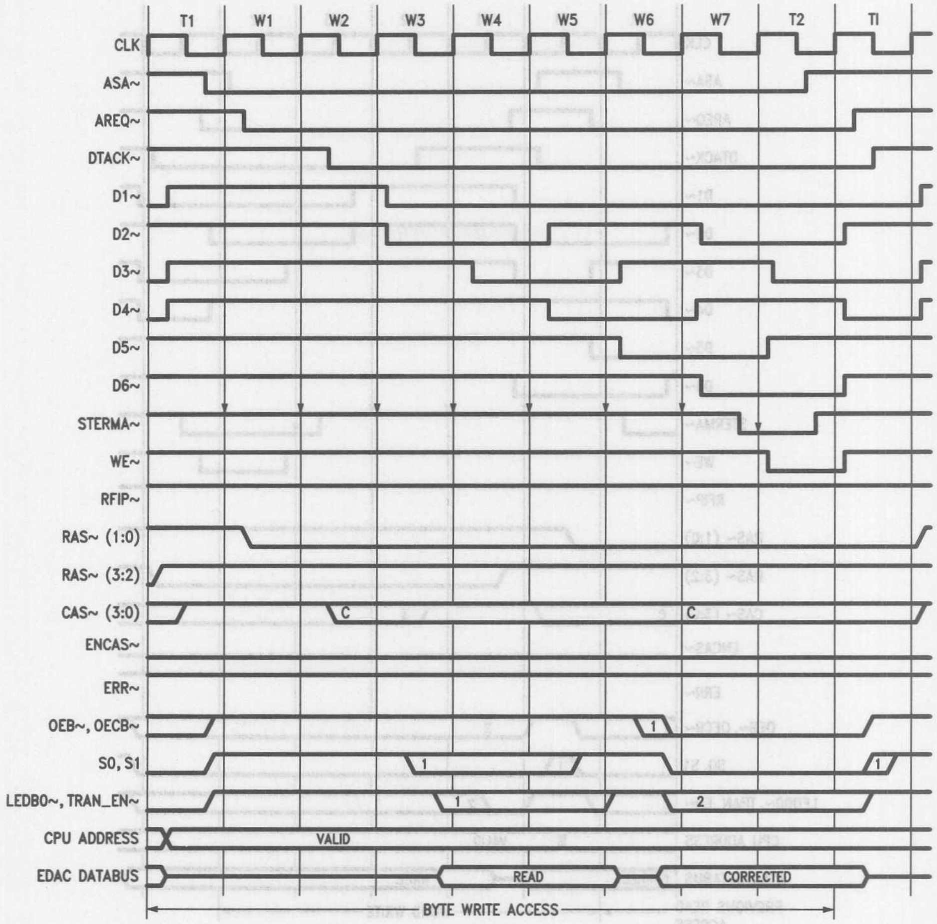
TL/F/9729-4

FIGURE 4. 68030 EDAC Burst Read Access Timing



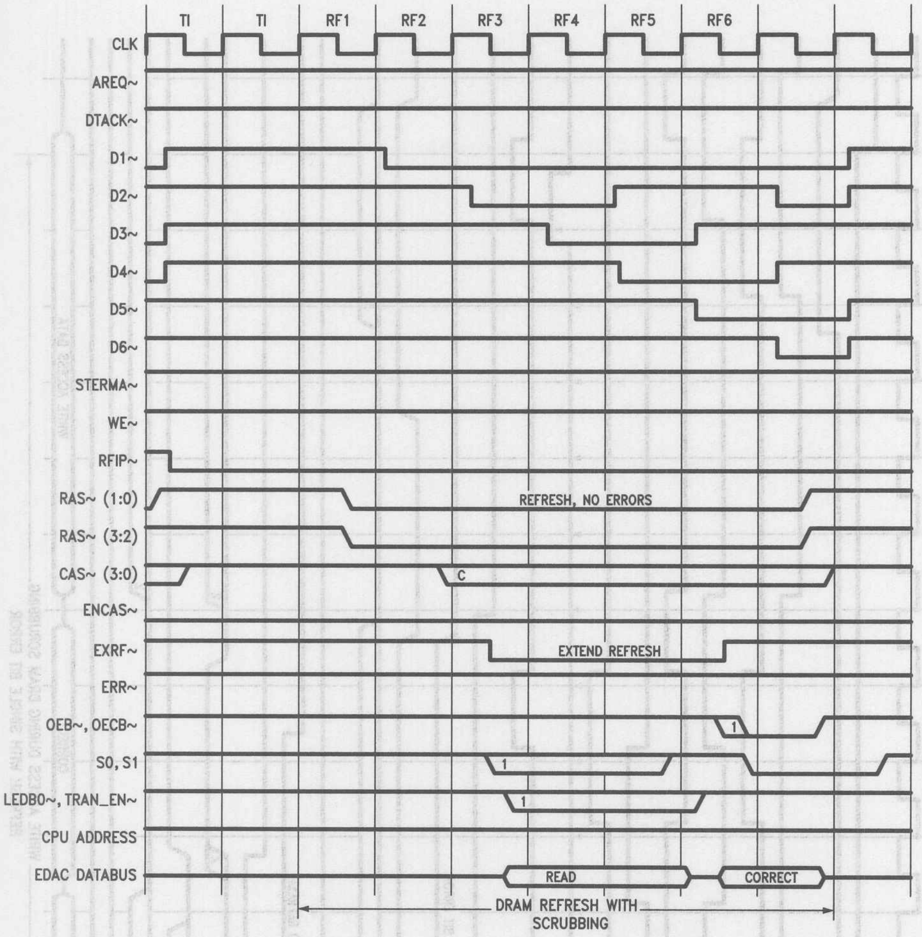
TL/F/9729-5

FIGURE 5. 68030 EDAC Word Write Access Timing



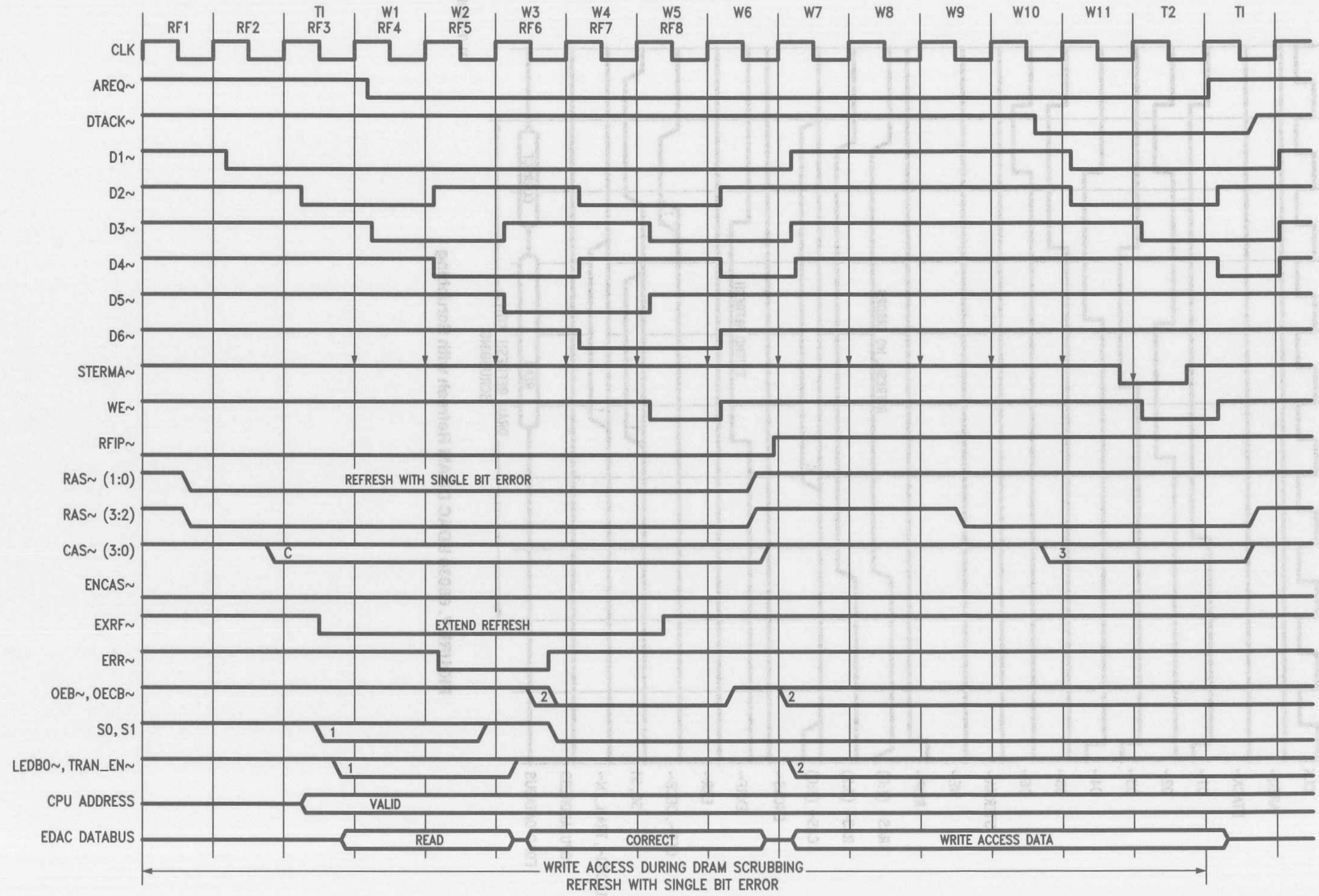
TL/F/9729-6

FIGURE 6. 68030 EDAC Byte Write Access Timing



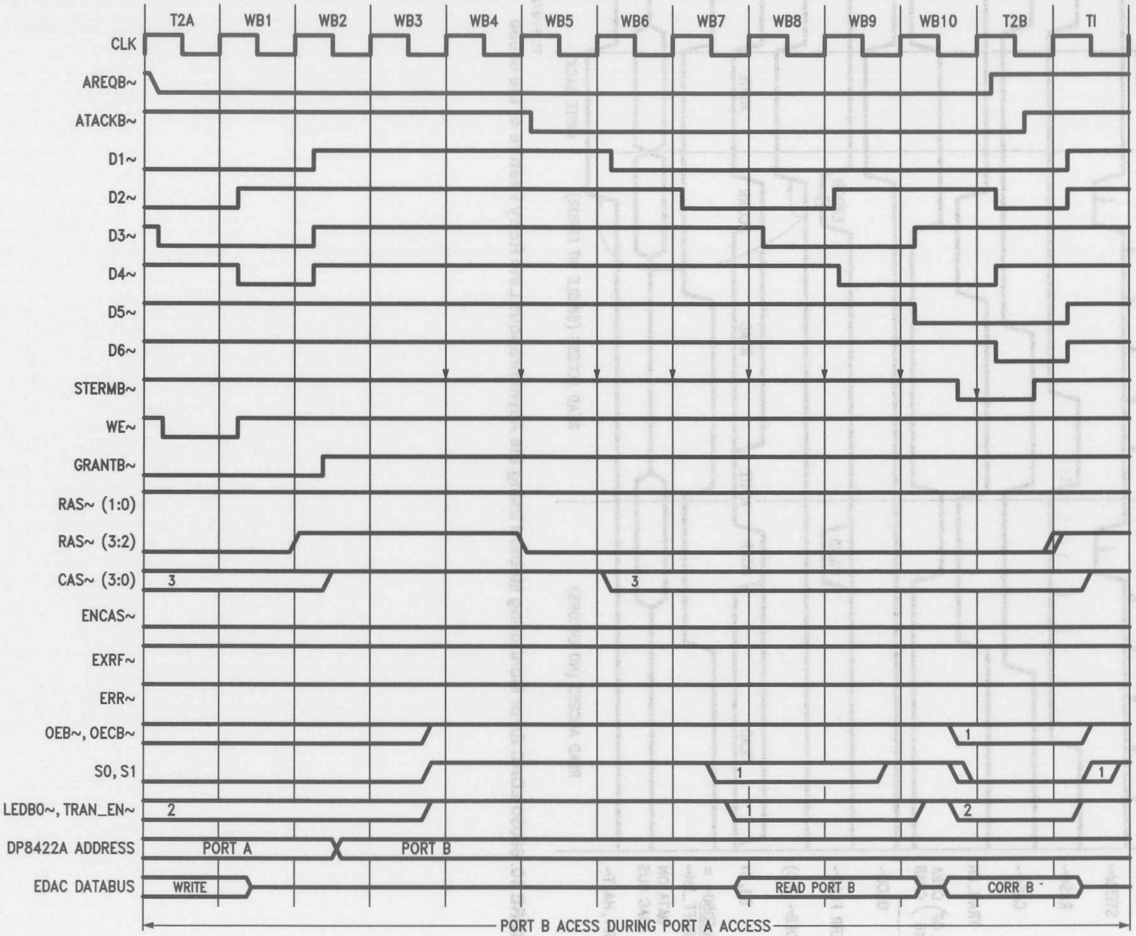
TL/F/9729-7

FIGURE 7. 68030 EDAC DRAM Refresh with Scrubbing



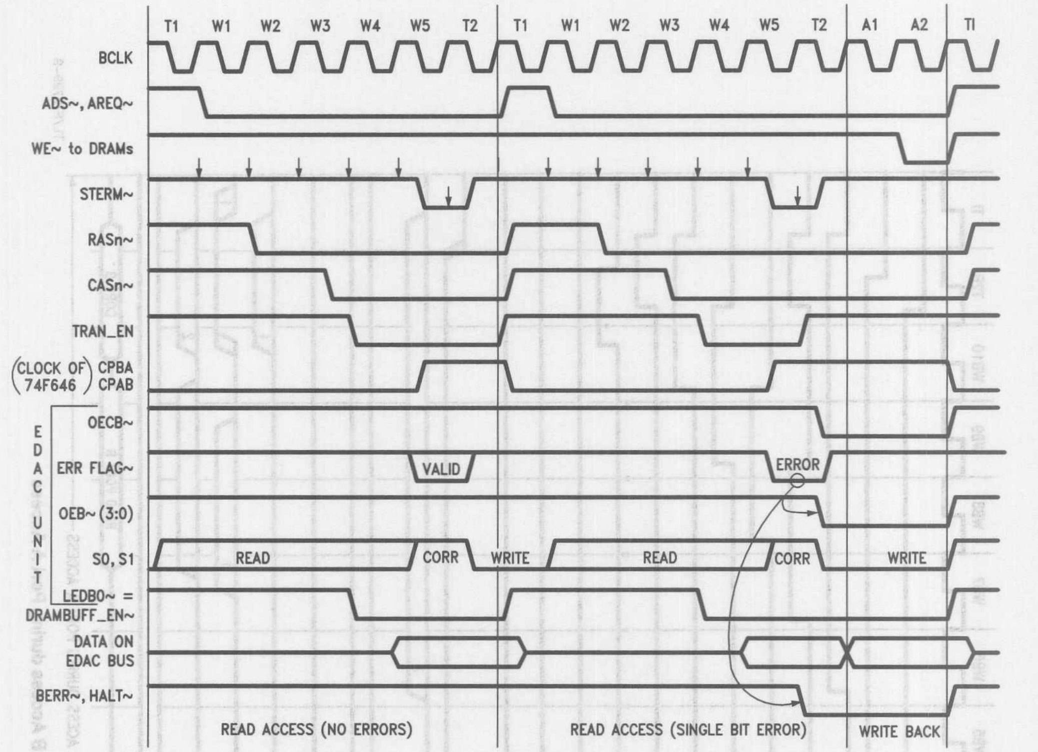
TL/F/9729-8

FIGURE 8. 68030 EDAC Write Access during Refresh Timing



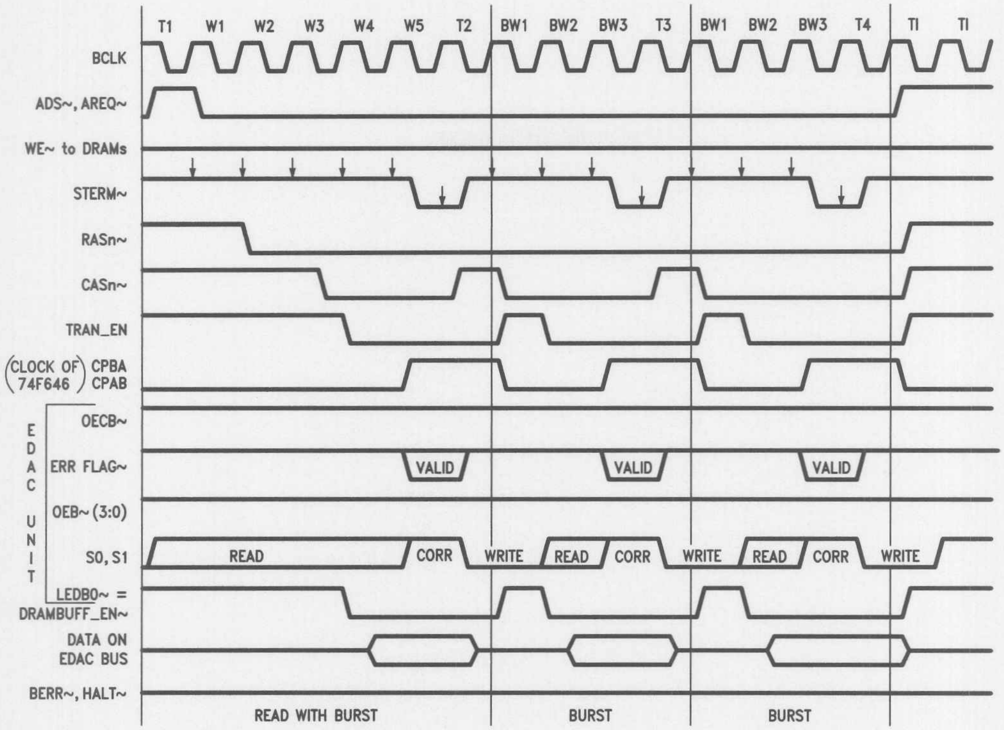
TL/F/9729-9

FIGURE 9. 68030 EDAC Port B Access during Port A Access



TL/F/9729-10

FIGURE 10. 68030 EDAC Error Monitoring Method Using the Asynchronous Late Retry Feature of the 68030



TL/F/9729–11

FIGURE 11. 68030 EDAC Error Monitoring Method Using the Asynchronous Late Retry Feature of the 68030

Section 3
High Density MAPL Family

Section 3 Contents

MAPL DATASHEETS

MAPL128/MAPL144 Multiple Array Programmable Logic 3-3
MAPL244 Multiple Array Programmable Logic (Preliminary) 3-15
MAPL268 Multiple Array Programmable Logic (Advanced Information) 3-31

MAPL APPLICATION EXAMPLES

1.0 MAPL Register Control 3-33
2.0 4-Bit Counter 3-38
3.0 3-Bit Up/Down Counter with 7-Segment Display Output 3-41
4.0 15-Bit Up/Down Counter 3-43
5.0 12-Bit L/R Shift Register 3-48
6.0 High Speed Frame Buffer Control 3-52
7.0 MAPL Divide Down Counter 3-60
8.0 Bus Transaction and DMA Controller 3-67
9.0 MAPL PC Interface Module for the FDDI MAC Layer Evaluation Board 3-78
10.0 PLUS405 to MAPL128 Conversion 3-85

MAPL APPLICATION NOTES

AN-810 Advanced PLD Architectures 3-87
AN-800 A Smart UART Design Implemented in a MAPL128 3-97
AN-801 State Machine Design 3-105
AN-802 MAPL Demonstration Board 3-116

National Semiconductor

MAPL128, MAPL144 Multiple Array Programmable Logic

General Description

The MAPL128 and MAPL144 are the first in a series of new higher density, electrically erasable CMOS (EECMOS) programmable logic devices based on a proprietary National Semiconductor programmable logic array architecture. The MAPL128 and MAPL144 products integrate multiple Field Programmable Logic Arrays (FPLAS), allowing for easy implementation of complex state machines, controllers, microinstruction sequencers, bus interfaces and general synchronous logic designs. The devices also allow for easy and cost effective integration of multiple TTL counter, register and logic functions.

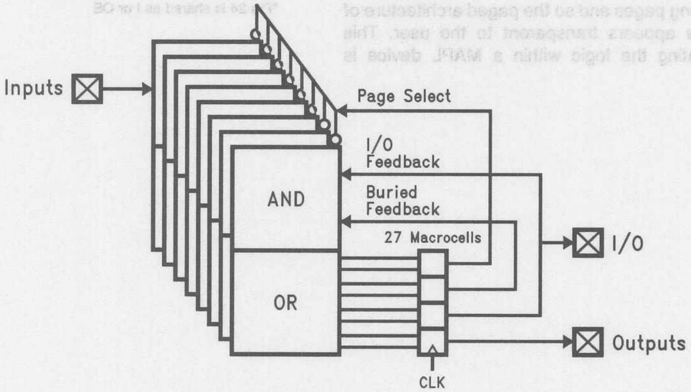
The MAPL™ product family is supported by National Semiconductor's OPAL™ (Open Programmable Architecture Language) software package and by other popular third-party PLD software packages such as ABEL™-4, Viewlogic®, CUPL™, LOG/ic and PLDesigner™. OPAL is an open environment design tool and is able to interface with these other software packages by means of the de-facto standard Berkeley PLA file format. OPAL also allows the designer to have full access to all of the intermediate files that are generated during the compilation process. Logic partitioning and utilization within a MAPL device may be done 100% automatically or manually using OPAL software. Automatic partitioning and device utilization can also be done using other third-party tools with the appropriate MAPL device fitter, available from the software vendor.

Programming of these devices is accomplished using readily available, industry standard, PLD programming equipment. NSC guarantees a minimum 100 erase/write cycles for all MAPL devices.

The MAPL128 and MAPL144 are available in 28-pin and 44-pin PLCC packages respectively. These packages conform to the JEDEC standard for power, ground and clock pin placements.

Features

- High density flexible PLA architecture
- 33, 40 or 45 MHz system performance
- Low power: I_{CC} = 110 mA max
- Electrically erasable CMOS technology:
  - — 100% functionally tested
  - — Instantly reconfigurable logic
  - — Minimum of 100 erase/write cycles
- 27 programmable macrocells with DE, JK, RS or T type registers
- Asynchronous preload, reset and power-up reset capability
- Replaces multiple PAL/PLA devices
- Fully supported by National's OPAL, and popular third-party development software
- Industry standard programmer support
- Security cell prevents copying



TL/L/11146-2

FIGURE 1. MAPL128

Functional and Architecture Description

The MAPL1 device architecture is functionally equivalent to a large continuous FPLA having a total of 128 product terms. In reality, the MAPL128 and MAPL144 each have eight individual FPLA planes or pages, consisting of both a programmable AND array as well as a programmable OR array, with the same 58 x 16 x 54 configuration. Each programmable AND array has a total of 58 true and complimentary inputs and 16 product terms which can be connected to any of the 54 sum terms in the programmable OR array.

Unlike PAL devices or complex PLD (CPLD) architectures based on PAL blocks, the product terms in an FPLA-based device can be shared among all outputs and macrocells since the OR array is programmable. Therefore, duplication of product terms for each output or state transition is not required.

The MAPL architecture provides 100% interconnectivity of the FPLA arrays. Any input can be routed to any output, via any one of the FPLA pages. There is no restriction on the number of product terms that can be connected to an output, no limiting switch matrix and no time consuming expander arrays to deal with as in the case of most PAL-based CPLDs.

Dynamic Logic Allocation (Paging)

Since sequencer and state machine designs require the use of only a portion of the total available logic at any given time, the MAPL device logic is partitioned into pages. Only the current page is active at any time, the remaining pages being effectively deactivated. Three buried page macrocells are used to select which one of the eight pages is active. In addition, just like the other buried macrocells, these page macrocells can be used to store state bits. Pages are selected on-the-fly. As the output macrocells are clocked, so too are the page macrocells. This has the effect of pre-selecting the page where the next state logic can be found. It may be the same page or it may be a different one. By pipelining the next page in this manner no time delay is incurred in switching pages and so the paged architecture of the MAPL device appears transparent to the user. This method of allocating the logic within a MAPL device is known as dynamic logic allocation or more simply paging. By using this technique, internal propagation delays and power consumption are kept to a minimum. In fact the MAPL1 series devices consume only as much power as a single standard EECMOS GAL device although they have several times the logic density.

To relieve the designer from having to fully understand the MAPL's paged architecture, a software fitter program (see development tools) is available. The fitter is responsible for automatically partitioning the logic and assigning the internal page bits and nodes, giving an immediate fit/no-fit result. Many other complex PLD fitters require the designer to design with device limitations in mind and to make repeated attempts to change and recompile the design to get it to fit into the target device.

TABLE II. MAPL128/144 Pin Description

Product	MAPL128	MAPL144
Power and GND Pins	2	8
Dedicated Clock	1	1
Dedicated Enable	0*	1
Setup Time Select Pin	—	1
Dedicated Inputs Pins	9*	9
Dedicated Outputs Pins	4	12
I/O Pins	12	12
Total Pins	28	44
Page Registers	3	3
Buried Registers	8	—
Total Output Registers	16	24
Total Registers	27	27
Total AND Array Inputs	32	32
Total OR Array Outputs	54	54
Logic Transition Terms	128	128
Control Terms	4	4
Sum Terms	432	432
Total Product and Sum Terms	564	564

*Pin 24 is shared as I or OE

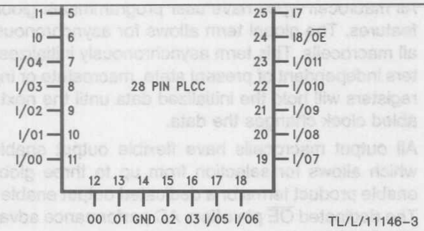


FIGURE 2. MAPL128 Pinout

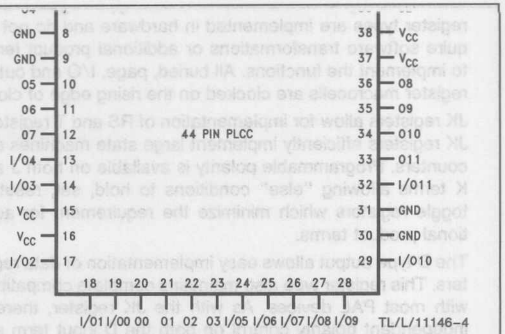
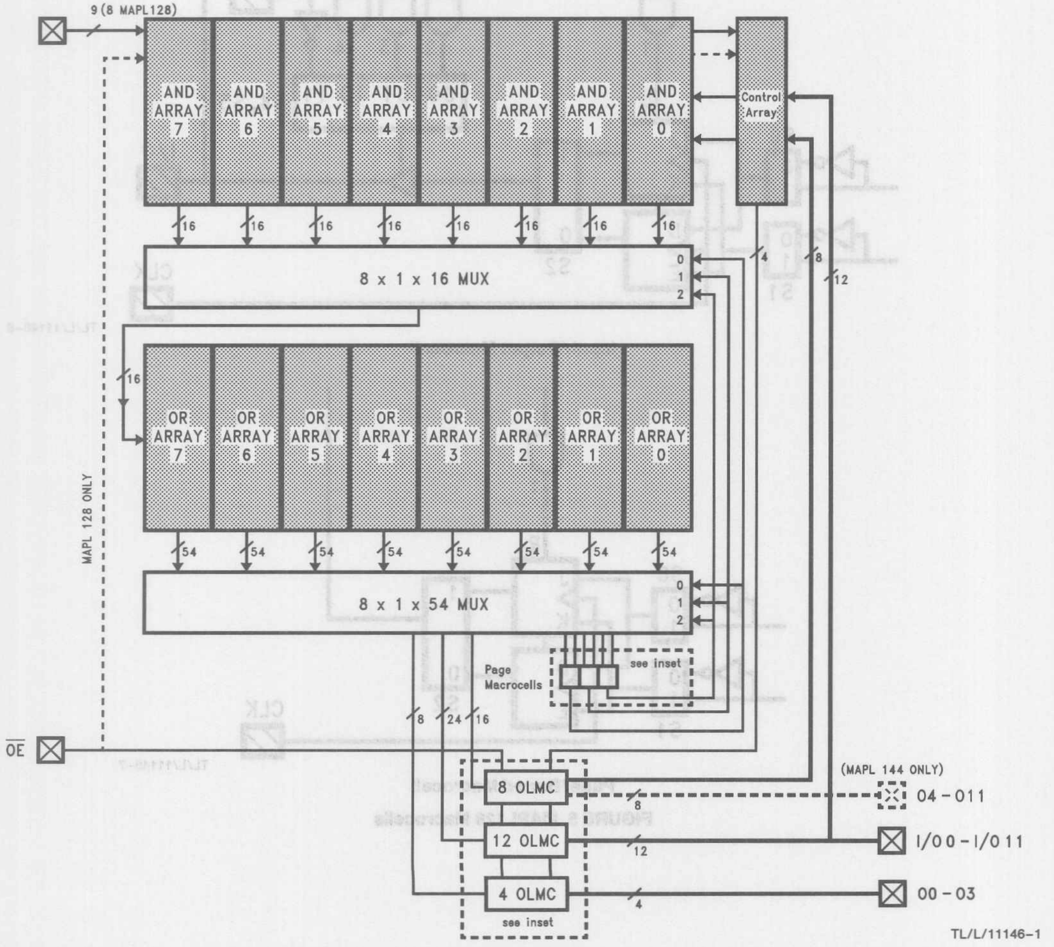


FIGURE 3. MAPL144 Pinout

MAPL128/144 Block Diagram



Logic Macrocell

The MAPL products include a flexible user configurable macrocell structure. The same registered macrocell is utilized for I/O, buried and output functions. Each macrocell can be configured as a true JK or DE type register. Both register types are implemented in hardware and do not require software transformations or additional product terms to implement the functions. All buried, page, I/O and output register macrocells are clocked on the rising edge of clock.

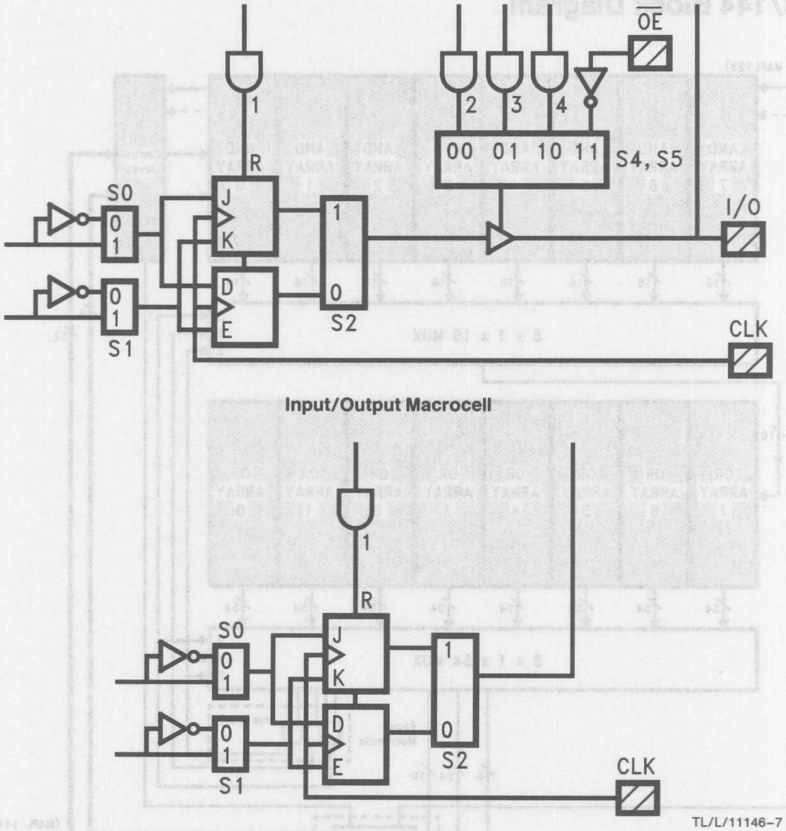
JK registers allow for implementation of RS and T registers. JK registers efficiently implement large state machines and counters. Programmable polarity is available on both J and K terms allowing "else" conditions to hold, set, reset or toggle registers which minimize the requirement for additional product terms.

The D type output allows easy implementation of data registers. This register type also maintains complete compatiblity with most PAL devices. As with the JK register, there is independent polarity control on both the D input term and the clock enable term. When the enable register input, E, is low, the register contents are unchanged regardless of CLK and D inputs.

All macrocell types have user programmable global control features. The global term allows for asynchronous reset of all macrocells. This term asynchronously initializes all registers independent of present state, macrostate or inputs. The registers will hold the initialized data until the next valid enabled clock changes the data.

All output macrocells have flexible output enable control which allows for selection from up to three global output enable product terms or a dedicated output enable ($\overline{\text{OE}}$) pin. The dedicated $\overline{\text{OE}}$ pin offers AC performance advantages in TRI-STATE® delays.

Each output register can be active low or active high. This allows I/O feedback or register output to be programmable active low or active high. This allows designs to have all registers type and both pin polarities.

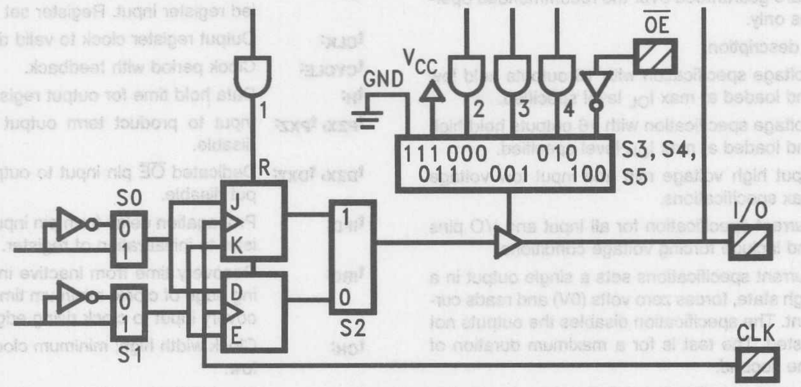


Input/Output Macrocell

Page/Buried Macrocell

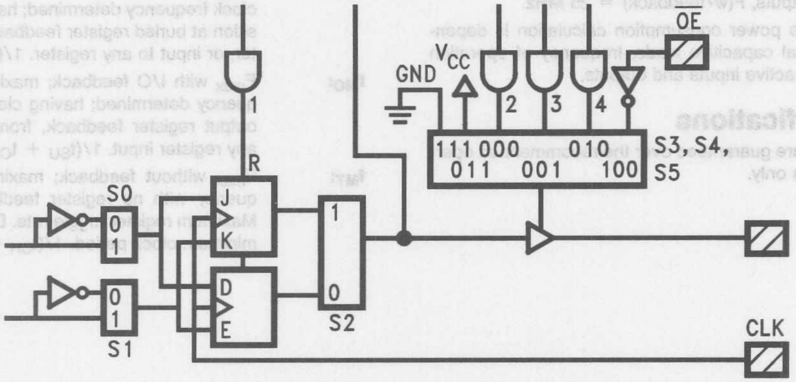
FIGURE 5. MAPL128 Macrocells

Logic Macrocell (Continued)



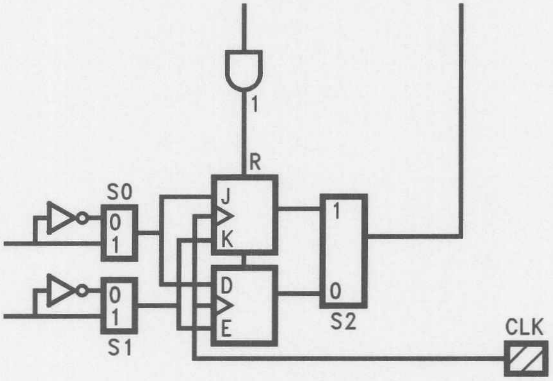
TL/L/11146-9

Input/Output Macrocell



TL/L/11146-10

Output/Buried Macrocell



TL/L/11146-11

Page Macrocell

FIGURE 6. MAPL144 Macrocells

DC Specifications

Specifications are guaranteed over the recommended operating conditions only.

DC parameter description:

V_{OL}: Voltage specification with 16 outputs held low and loaded at max I_{OL} level specified.

V_{OH}: Voltage specification with 16 outputs held high and loaded at max I_{OH} level specified.

V_{IH}, V_{IL}: Input high voltage min and input low voltage max specifications.

I_{IH}, I_{IL}: Current specification for all input and I/O pins and include forcing voltage conditions.

$I_{OS}(I_O)$: Current specifications sets a single output in a high state, forces zero volts (0V) and reads current. The specification disables the outputs not tested. The test is for a maximum duration of one second.

I_{CC}: Current specification includes V_{CC}, temp, frequency of operation, number of outputs active and output loading. I_{CC} specified with open outputs, F(w/feedback) = 25 MHz.

The worst case power consumption calculation is dependent on external capacitive loads, frequency of operation and number of active inputs and outputs.

AC Specifications

Specifications are guaranteed over the recommended operating conditions only.

AC parameter description:

t_{SU}: Data input or buried feedback to output or buried register input. Register set up time.

t_{CLK}: Output register clock to valid data output.

t_{CYCLE}: Clock period with feedback.

t_H: Data hold time for output register clock.

t_{PZX}, t_{PXZ}: Input to product term output enable, output disable.

t_{DZX}, t_{DXZ}: Dedicated $\overline{OE}$ pin input to output enable, output disable.

t_{IPD}: Propagation delay from pin input or buried register to initialization of register.

t_{IRC}: Recovery time from inactive input to valid rising edge of clock; minimum time from reset recovery input to clock rising edge.

t_{CH}: Clock width high; minimum clock high to clock low.

t_{CL}: Clock with low. Minimum clock low to clock high.

f_{MB}: F_{max} with buried feedback or input. Maximum clock frequency determined; having clock transition at buried register feedback, to any register, or input to any register. $1/(t_{SU})$

f_{MO}: F_{max} with I/O feedback; maximum clock frequency determined; having clock transition at output register feedback, from output pin to any register input. $1/(t_{SU} + t_{CO})$

f_{MT}: F_{max} without feedback; maximum clock frequency with no register feedback or inputs. Maximum register toggle rate. Determined with minimum clock period. $1/(t_{CH} + t_{CL})$

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C
ESD Tolerance	750V

C_{ZAP} = 100 pF
R_{ZAP} = 150Ω
Test Method: Human Body Model
Test Specification: NSC SOP-5-028 REV.C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		V_{CC}+1	V
V_{IL}	Low Level Input Voltage				−1.0		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA	COM/IND	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 8 mA (128)	COM/IND			0.5	V
			I_{OL} = 12 mA (144)					
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−160	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max	128	COM			110	mA
			144	COM			125	mA
				IND			130	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					10	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					12	pF

*One output at a time for a maximum duration of one second.

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Switching Characteristics Over Recommended Operating Conditions (Note)

AC TIMING REQUIREMENTS

Symbol	Parameter		Conditions	MAPL144-45		MAPL128/144-40		MAPL128/144-33		Units
				COM		COM		COM		
				Min	Max	Min	Max	Min	Max	
t_{SU}	Set-Up Time (Input or Feedback before Clock)		MAPL128	—		17		20		ns
			MAPL144, Pin 44 Low[3]	15		15		18		ns
			MAPL144, Pin 44 High[3]	12		12		15		ns
t_H	Hold Time (Input after Clock)		MAPL128	0		0		0		ns
			MAPL144, Pin 44 Low[3]	0		0		0		ns
			MAPL144, Pin 44 High[3]	2		2		2		ns
t_W	Clock Pulse Width (High/Low)			8		8		8		ns
t_{CYCLE}	Clock Cycle Period (with Buried Feedback)			22		25		30		ns
f_{MB}	Clock Frequency	with Buried Feedback[1]			45.5		40.0		33.3	MHz
f_{MO}[2]		with I/O Feedback[1]			40.0		40.0		33.3	MHz
f_{MT}		without Feedback			62.5		62.5		62.5	MHz
t_{PR}	Clock Valid after Power-Up				100		100		100	ns
t_{CLK}[2]	Clock Input to Registered Output or Feedback		MAPL128		—		9		10	ns
			MAPL144, Pin 44 Low[3]		10		11		12	ns
			MAPL144, Pin 44 High[3]		13		14		15	ns
t_{PZXG}	$\overline{OE}$ ↓ to Registered Output Enabled		Active High; S1 Open, $C_L = 35$ pF Active Low; S1 Closed, $C_L = 35$ pF		10		10		12	ns
t_{PXZG}	$\overline{OE}$ ↑ to Registered Output Disabled		From V_{OL}; S1 Open, $C_L = 5$ pF From V_{OH}; S1 Closed, $C_L = 5$ pF		10		10		12	ns
t_{PZXI}	Input to Output Enable via Product Term		Active High; S1 Open, $C_L = 35$ pF Active Low; S1 Closed, $C_L = 35$ pF		15		15		18	ns
t_{PXZI}	Input to Output Disabled via Product Term		From V_{OL}; S1 Open, $C_L = 5$ pF From V_{OH}; S1 Closed, $C_L = 5$ pF		15		15		18	ns
t_{RESET}	Power Up to Registered Output Low		S1 Closed, $C_L = 35$ pF		45		45		45	µs
t_{IPD}	Input Initialization to Output Reset				18		18		20	ns
t_{IRC}	Input Initialization before Clock				12		15		20	ns

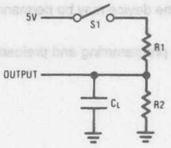
[1] Actual measurements

[2] S1 CLOSED, $C_L = 35$ pF

[3] The sp pin (44) is used to select t_{SU}/t_{CLK}

Note: All MAPL144 specifications are Preliminary.

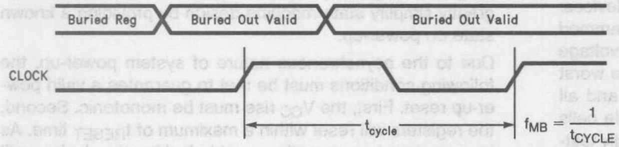
AC Test Load (All AC specifications are measured with half the outputs switching)



MAPL128	MAPL144
R1 = 620Ω	R1 = 400Ω
R2 = 620Ω	R1 = 400Ω

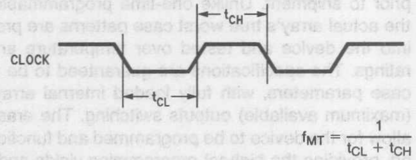
TL/L11146-23

Timing Diagrams



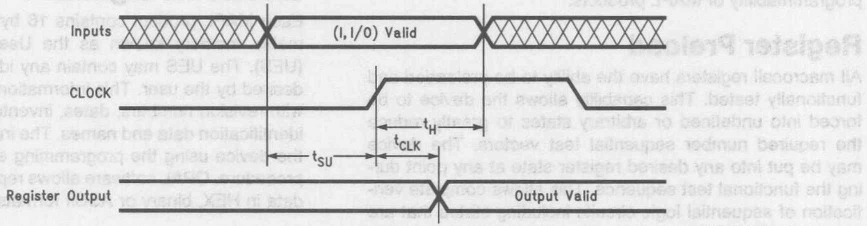
TL/L/11146-12

FIGURE 8. Internal Feedback Frequency



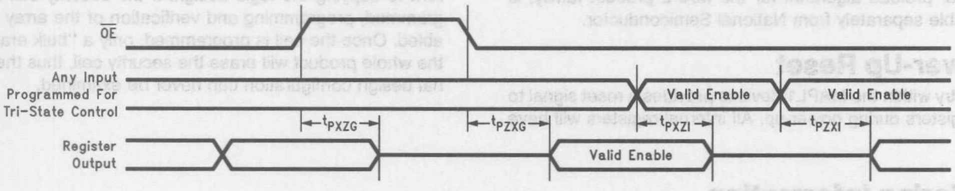
TL/L/11146-13

FIGURE 9. Toggle Frequency



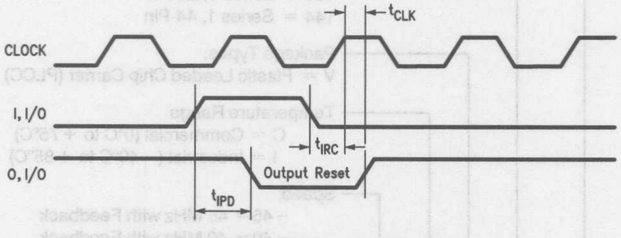
TL/L/11146-14

FIGURE 10. External Feedback



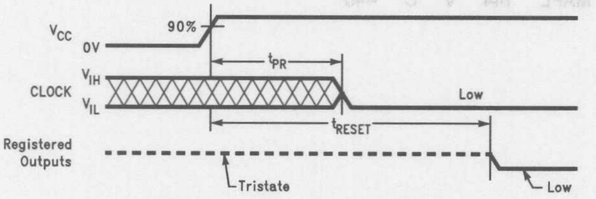
TL/L/11146-15

FIGURE 11. Output Enable Timing



TL/L/11146-16

FIGURE 12. Initialization Timing



TL/L/11146-17

FIGURE 13. Power-Up Reset Timing

Design Technology Testability

Electrically Erasable EECMOS technology allows the MAPL product family to be 100% AC, DC and functionally tested prior to shipment. Unlike one-time programmable devices, the actual array's true worst case patterns are programmed into the device and tested over temperature and voltage ratings. The specifications are guaranteed to be true worst case parameters, with fully loaded internal arrays and all (maximum available) outputs switching. The erasable cells allow for the device to be programmed and functionally tested, providing the highest programming yields and post-programming, functional yields, available in a user programmable device. National Semiconductor guarantees 100% field programmability of MAPL products.

Register Preload

All macrocell registers have the ability to be preloaded and functionally tested. This capability allows the device to be forced into undefined or arbitrary states to greatly reduce the required number sequential test vectors. The device may be put into any desired register state at any point during the functional test sequence. This allows complete verification of sequential logic circuits including states that are not defined in the normal operational state sequence. The register preload is not an operational mode. It is available in certified programming equipment for test purposes. The register preload algorithm for the MAPL product family, is available separately from National Semiconductor.

Power-Up Reset

Circuitry within the MAPL1 devices provides a reset signal to all registers during power-up. All internal registers will have their Q outputs set low after a specified time (t_{RESET}). As a result, the state on the registered output pins (if they are enabled) will always be low on power-up, regardless of the programmed polarity of the output pins. This feature can greatly simplify state machine design by providing a known state on power-up.

Due to the asynchronous nature of system power-up, the following conditions must be met to guarantee a valid power-up reset. First, the V_{CC} rise must be monotonic. Second, the registers will reset within a maximum of t_{RESET} time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

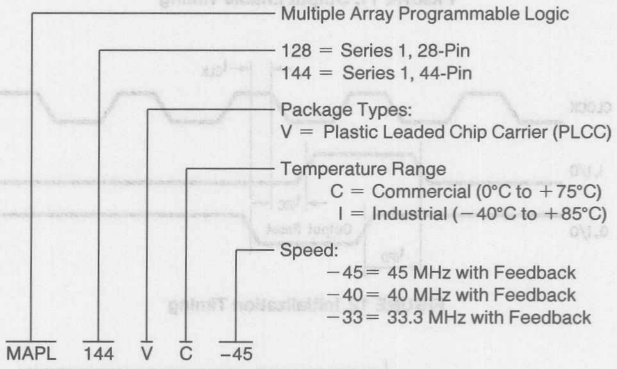
Electronic Signature

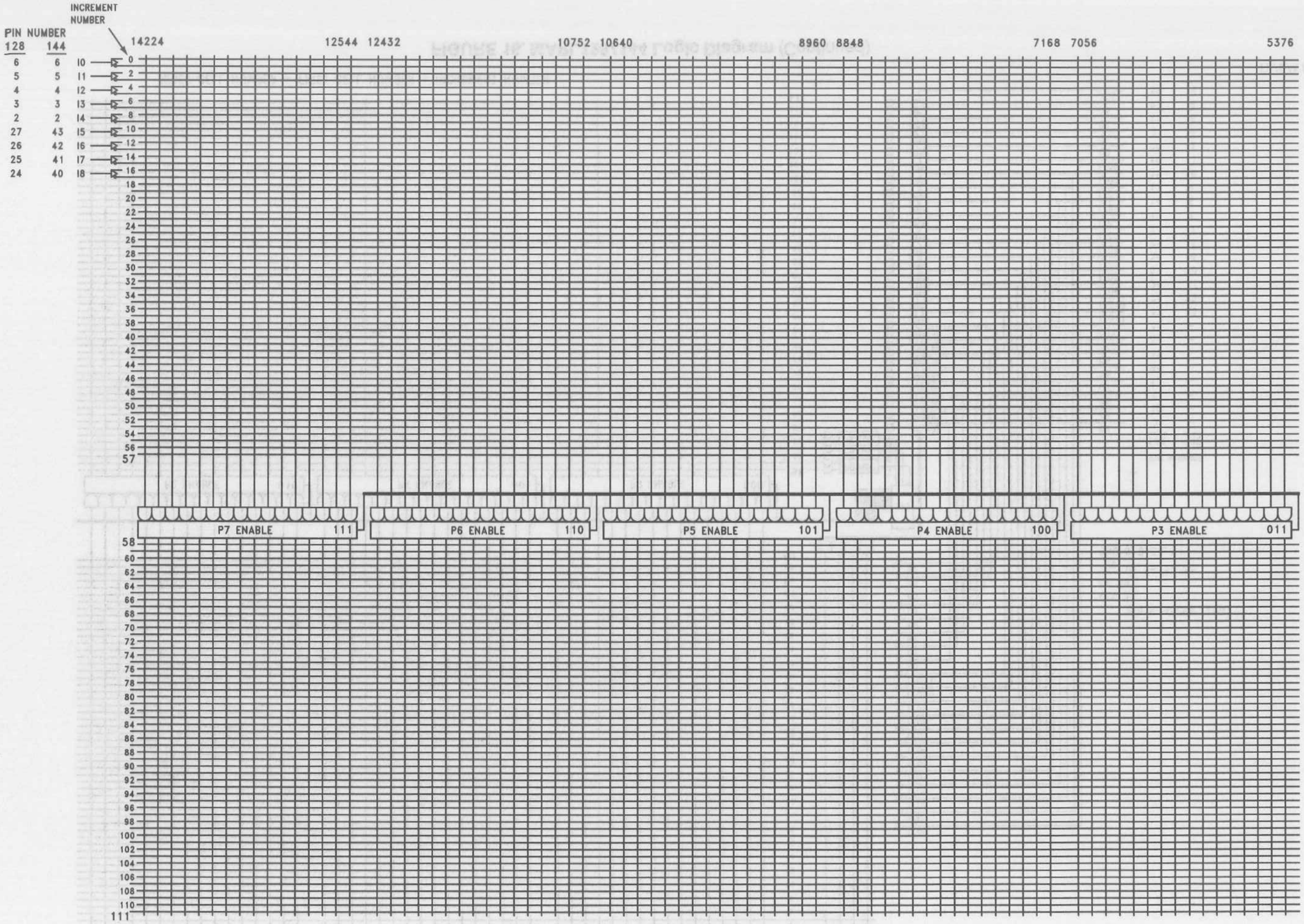
Each MAPL product contains 16 bytes of user reprogrammable memory known as the User Electronic Signature (UES). The UES may contain any identification information desired by the user. This information typically is associated with revision numbers, dates, inventory control information, identification data and names. The information is read out of the device using the programming equipment's verification procedure. OPAL software allows representation of the UES data in HEX, binary or ASCII formats.

Design Security

A security cell is provided on all MAPL products as a deterrent to copying the logic design. If the security cell is programmed, programming and verification of the array is disabled. Once the cell is programmed, only a "bulk erase" of the whole product will erase the security cell, thus the original design configuration can never be examined.

Ordering Information

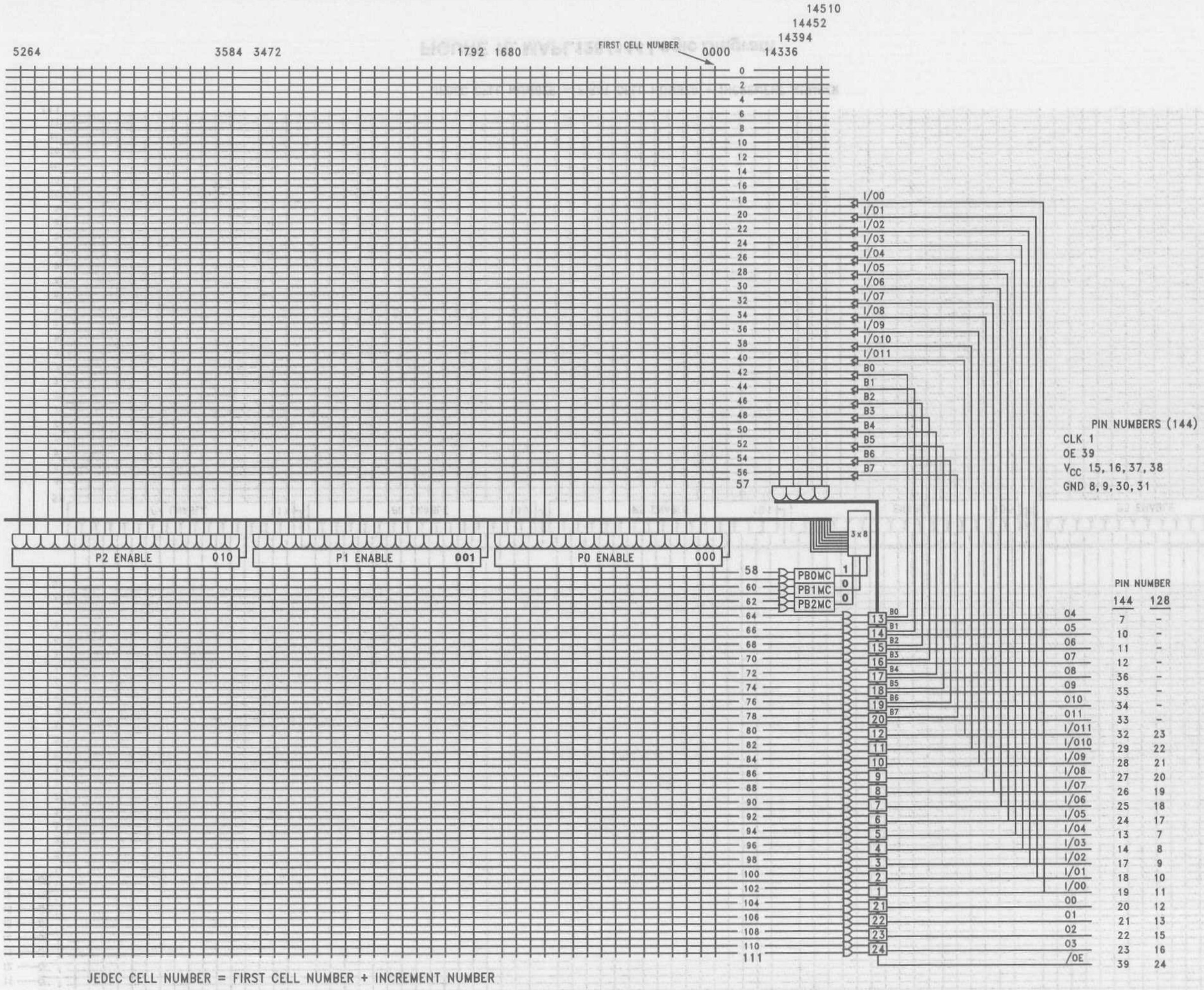

Multiple Array Programmable Logic
128 = Series 1, 28-Pin
144 = Series 1, 44-Pin
Package Types:
V = Plastic Leaded Chip Carrier (PLCC)
Temperature Range
C = Commercial (0°C to +75°C)
I = Industrial (−40°C to +85°C)
Speed:
−45 = 45 MHz with Feedback
−40 = 40 MHz with Feedback
−33 = 33.3 MHz with Feedback
MAPL
144
V
C
−45



JEDEC CELL NUMBER = FIRST CELL NUMBER + INCREMENT NUMBER

TL/L/11146–20

FIGURE 16. MAPL128/144 Logic Diagram



TL/L/11146–21

FIGURE 16. MAPL128/144 Logic Diagram (Continued)

MAPL244 Multiple Array Programmable Logic

General Description

The MAPL244 is a second generation device in a series of new higher density, electrically erasable CMOS (EECMOS) programmable logic devices based on a proprietary National Semiconductor programmable logic array architecture. The MAPL244 integrates multiple Field Programmable Logic Arrays (FPLA), allowing for easy implementation of complex state machines, controllers, microinstruction sequencers, bus interfaces and general synchronous logic designs. In addition to the MAPL1xx devices the MAPL244 incorporates a separate PAL® block with combinatorial I/O to handle asynchronous and decode functions. The MAPL244 allows for easy and cost effective integration of complex subsystems consisting of synchronous and asynchronous elements.

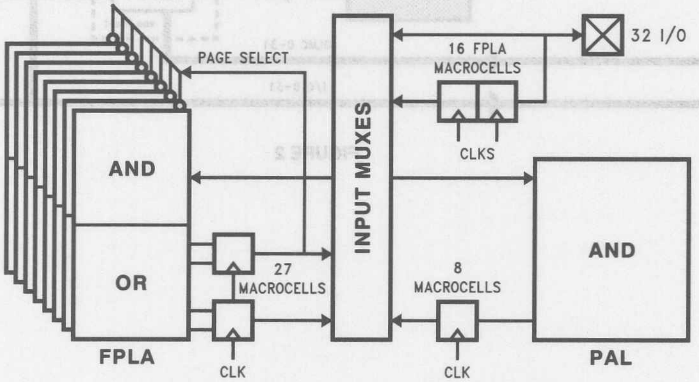
The MAPL™ product family is supported by National Semiconductor's OPAL™ (Open Programmable Architecture Language) software package and by other popular third-party PLD software packages such as ABEL-4, CUPL, PLDesigner and Viewlogic®. OPAL is an open environment design tool and is able to interface with these other software packages by means of the de-facto standard Berkeley PLA file format. OPAL also allows the designer to have full access to all of the intermediate files that are generated during the compilation process. Logic partitioning and utilization within a MAPL device may be done 100% automatically or manually using OPAL software. Automatic partitioning and device utilization can also be done using other third-party tools with the appropriate MAPL device fitter, available from the software vendor.

Programming MAPL devices is accomplished using readily available, industry standard, PLD programming equipment. NSC guarantees a minimum 100 erase/write cycles for all MAPL devices.

The MAPL244 is available in a 44-pin PLCC package. It conforms to the JEDEC standard for power, ground and clock pin placements.

Features

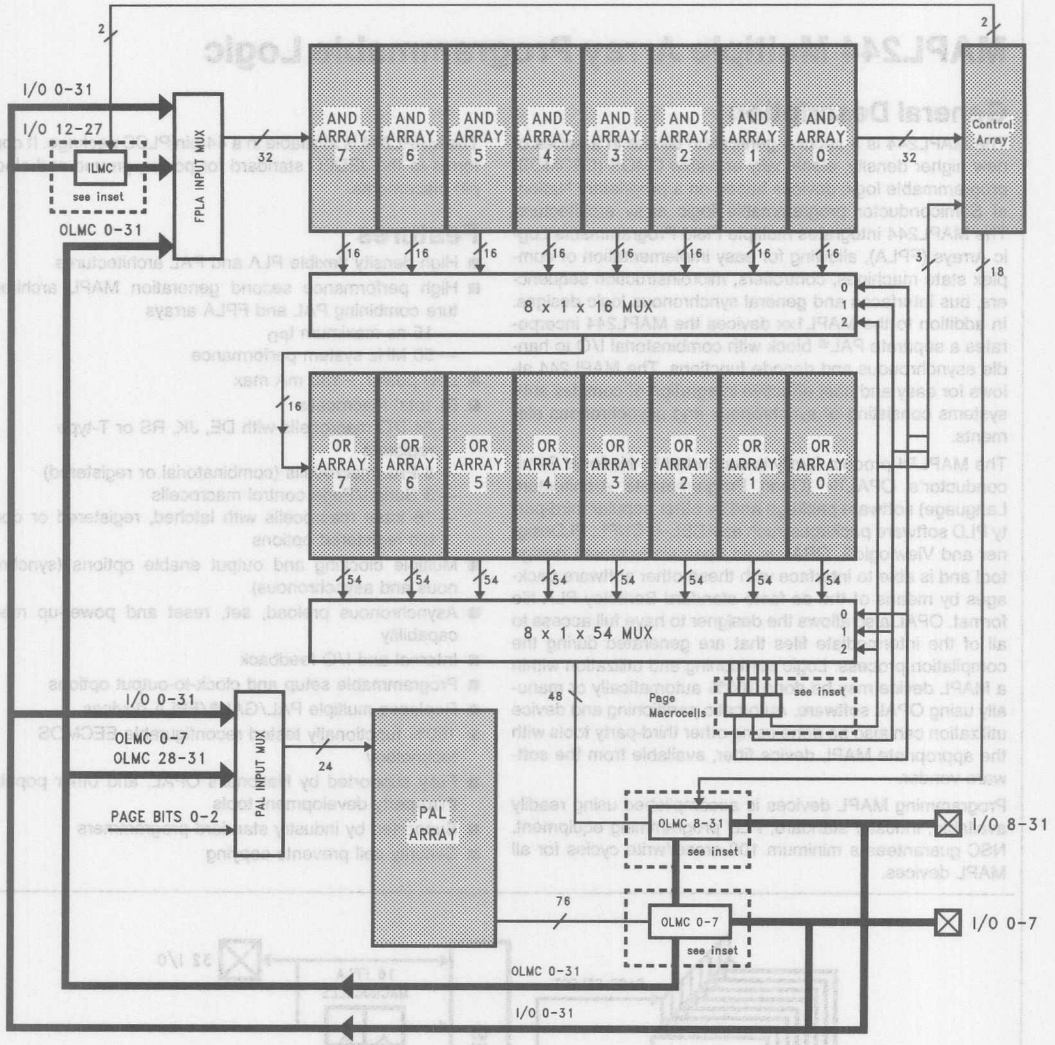
- High density flexible PLA and PAL architectures
- High performance second generation MAPL architecture combining PAL and FPLA arrays
  - — 15 ns maximum t_{PD}
  - — 50 MHz system performance
- Low power −180 mA max
- 51 total macrocells
  - — 24 I/O macrocells with DE, JK, RS or T-type registers
  - — 8 PAL macrocells (combinatorial or registered)
  - — 3 Buried/Page control macrocells
  - — 16 input macrocells with latched, registered or double registered options
- Multiple clocking and output enable options (synchronous and asynchronous)
- Asynchronous preload, set, reset and power-up reset capability
- Internal and I/O feedback
- Programmable setup and clock-to-output options
- Replaces multiple PAL/GAL®/FPLA devices
- 100% functionally tested reconfigurable EECMOS technology
- Fully supported by National's OPAL, and other popular third-party development tools
- Supported by industry standard programmers
- Security cell prevents copying



TL/H/11324-1

FIGURE 1. MAPL244

MAPL244 Block Diagram



TL/H/11324-2

FIGURE 2

MAPL244 Functional Description

The MAPL244 features an FPLA array similar to the MAPL1 series devices and an additional PAL (programmable AND) and control array. There are eight individual pages in the FPLA array, each now has a 64 x 16 x 54 configuration (64 input lines, 16 product terms and 54 output lines) giving a total of 128 product terms, feeding 27 macrocells. The PAL array has a 48 x 76 configuration (48 input lines and 76 product terms) which feed eight additional I/O logic macrocells (IOLMCs). There are a total of 35 IOLMCs in the MAPL244 *(*see *Figure 2*). The eight IOLMCs in the PAL array are individually configurable to provide registered or combinational outputs, with programmable polarity. Alternatively they may provide a buried feedback path and the I/O pins may be used as dedicated inputs. There is a choice of clocks, output enable and reset conditions.

The MAPL244 incorporates a flexible macrocell architecture that implements both DE-type and JK registers, with clock enables. This alleviates the need for software transformations or additional product terms to implement hold and toggle functions.

Unlike PAL devices, or complex PLD (CPLD) architectures based on PAL blocks, the product terms in the FPLA-based MAPL244 device can be shared among all outputs and macrocells since the OR array is user programmable. Therefore duplication of product terms for each output or state transition is not required. The multiple FPLA array within the MAPL244 provides 100% interconnectivity of the FPLA arrays. Any input can be routed to any output within the array. There is no restriction on the number of product terms that can be connected to an output, no limiting switch matrix and no time consuming expander arrays to deal with as in the case of most PAL-based CPLDs.

Dynamic Logic Allocation (Paging)

Since sequencer and state machine designs require the use of only a portion of the total available logic at any given time, the FPLA array of the MAPL device is partitioned into pages. Only the current page is active at any time, the remaining pages being effectively deactivated. The additional PAL array and control array of the MAPL244 are always active and available. Three buried page macrocells are used to select which one of the eight FPLA pages is active. In addition, just like the other buried macrocells, these page macrocells can be used to store state bits. Pages are enabled on-the-fly. As the output macrocells are clocked, so too are the page macrocells. This has the effect of pre-selecting the page where the next state logic can be found. It may be the same page or it may be a different one. By pipelining the next page in this manner no time delay is incurred in switching pages and so the paged architecture of the MAPL device appears transparent to the user. This method of allocating the logic within a MAPL device is known as dynamic logic allocation or more simply paging. By using this technique, internal propagation delays and power consumption are kept to a minimum. The MAPL1 series devices consume only as much power as a single standard EECMOS GAL device, and the second generation MAPL244 device only takes as much power as a solitary 20-pin bipolar PAL device, although both MAPL series devices have several times the logic density of each respectively.

To relieve the designer from having to fully understand the MAPL's paged architecture, a software fitter program (see MAPL244 Development Tools) is available. The fitter is responsible for automatically partitioning the logic and assigning the internal page bits and nodes, giving an immediate fit/no-fit result.

Input Muxes

Signals coming from package pins, buried feedback, and ILMC's form the inputs to the PLA and PAL arrays. Since only a portion of these signals are needed as array inputs, 56 independent programmable multiplexers (40 2-to-1 and 16 3-to-1) are used to route selected signals to the inputs of the arrays. By making these multiplexers small and by providing many of them, great flexibility is achieved. For instance, most inputs are available to both the PLA and the PAL array simultaneously. Additionally, all I/O can be used as both inputs and buried feedback, allowing the designer to use I/O as inputs without sacrificing buried registers. The possible routing combinations are listed below.

MAPL244 I/O	Available ILMC	Pin Input to	Output from	Buried Feedback to
I/O 0–7	NO	PLA and PAL	PAL	PLA and PAL
I/O 8–11	NO	PLA and PAL	PLA	PLA
I/O 12–19	YES*	PLA and PAL	PLA	PLA
I/O 20–27	YES*	PLA and PAL	PLA	PLA
I/O 28–31	NO	PLA and PAL	PLA	PLA and PAL
Page Registers 0–2	N/A	N/A	PLA	PLA and PAL

*The 16 ILMC's are available to the PLA only.

Input Muxes (Continued)

It is important to again recognize that routing an input to the PLA doesn't preclude the routing of the same signal to the PAL, nor does routing buried feedback to one array preclude the routing of the feedback to the other array. In fact, it can be seen from the table that some I/Os allow all four combinations simultaneously.

While the designer can manually define how the signals are routed, device fitter software can automatically make these determinations based on the design information.

Control Array

In addition to the PLA and PAL arrays, the MAPL2 devices contain a 70 x 20 control array. This array provides output enable and initialization control to the PLA output macrocells. The macrocells are grouped into three sets, with each set having two output enable signals one and three product terms. The macrocells can be individually configured to use either of these output enable signals, V_{CC}, dedicated output enable pin, or their complements.

The control array also provides four initialization signals to the PLA output macrocells. One signal has three product terms, and the others have one product term. The macrocells can be individually configured to use these initialization signals and their complements to perform set and reset functions upon the macrocell registers.

The Input Logic Macrocells (ILMC's) have a one product term initialization signal and a one product term asynchronous clock signal. The ILMC initialization signal is separate from the output macrocell initialization signals so that the output and input macrocells can be initialized separately. The ILMC asynchronous clock signal allows the designer to asynchronously latch or buffer system data.

Note that the outputs of the page macrocells and their complements are also available in the control array to provide initialization and output enables based on state, since the page bits can also be used as state bits.

Input Logic Macrocell (ILMC)

The MAPL2 devices provide 16 input logic macrocells that can each be configured as a latch, register, or double register. The latch and the first register can be clocked by the system clock pin, an input clock pin or its complement, or an asynchronous (product term) clock. The second register is clocked by the system clock pin. By using the ILMC, the designer is able to better accommodate the setup and hold times of the device and the surrounding environment, thereby improving metastability. For instance, a particular system may only guarantee data on a bus for 5 ns. Thus, a PLD with a setup time >5 ns will not read this data reliably. Using the ILMC, the data can be latched without violating the ILMC's setup time. Now that the data has been acquired, it needs to be available to the PLA at the same time as other forms of input (other inputs, buried feedback, etc.). To guarantee that all of these signals are available to the PLA at once, a designer may instead choose to use a double registered approach. This way, the first register can successfully acquire the system data, and the second register can provide the data to the PLA, synchronizing the data flow between the system and the IC. In either case, the ILMC can be useful in meeting the timing requirements of the application.

Note that the ILMC may also be bypassed if a buffered input is not required (see table under "Input Muxes").

Setup/Clock-to-Output Time Selection

MAPL2 devices allow the designer to choose from a pair of setup and clock to output times to best meet individual system requirements. This speed option is software programmable by selecting the option when running the MAPL2 device fitter (see the OPAL manual or the device fitter software manual).

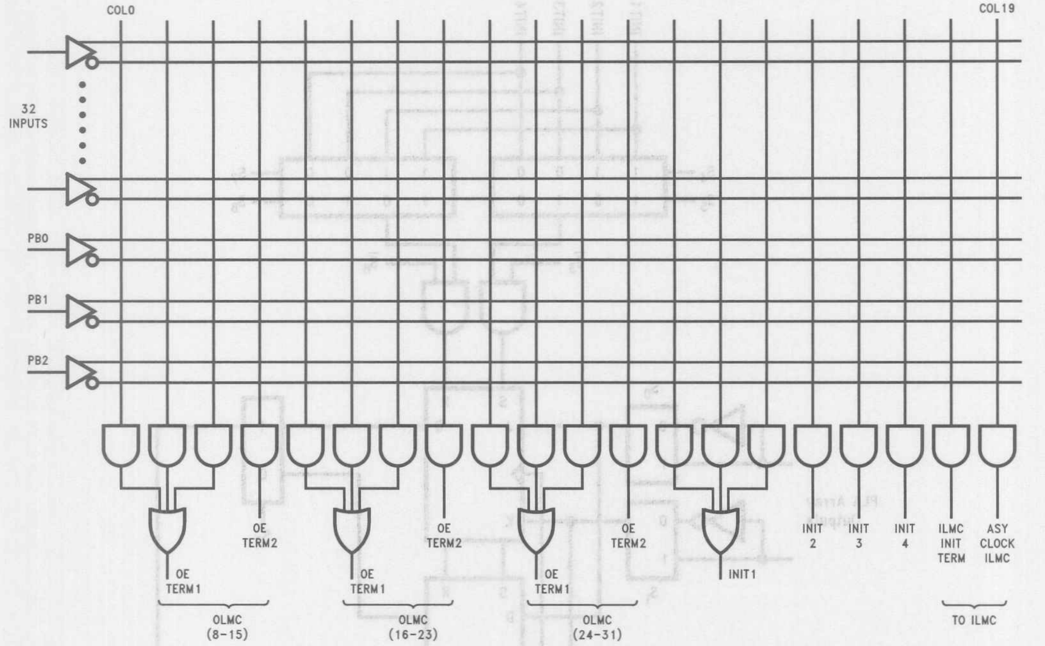
Output Logic Macrocells (OLMC's)

MAPL2 devices have three types of OLMC's: PLA, PAL, and page OLMC's. The PLA and page macrocells implement both DE and JK flip-flops in hardware with separate J/D and K/E product terms, allowing the designer to realize DE, JK, RS, T and D flip-flops with equal performance. DE and JK flip-flops are important in state machine design where the enable signal of a DE flip-flop allows the output to be held without multiple product terms, and where the JK flip-flops can be used to implement if-then-else statements or counters with a minimum number of product terms. The page OLMC's share this functionality with the PLA OLMC's since page macrocells can also be used as buried state bits.

The PAL OLMC is similar to a standard GAL macrocell and implements a D flip-flop in hardware. In addition, the PAL OLMC can be clocked by the system clock pin, a dedicated PAL clock pin, or an asynchronous (product term) clock. This clocking flexibility helps the designer synchronize the PAL with the PLA and the rest of the system. The register can be bypassed to implement combinatorial functions.

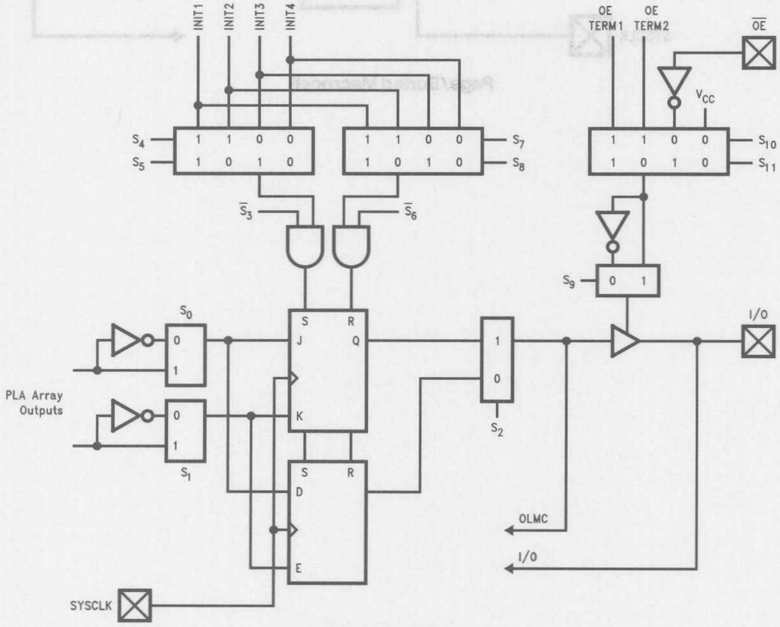
The PLA and the PAL OLMC's provide fixed, product term, and dedicated pin output enables (and their complements). In addition, these OLMC's can be initialized (set or reset) via a number of initialization product terms (see control array description).

FPLA Control Array (70 x 20)



TL/H/11324-4

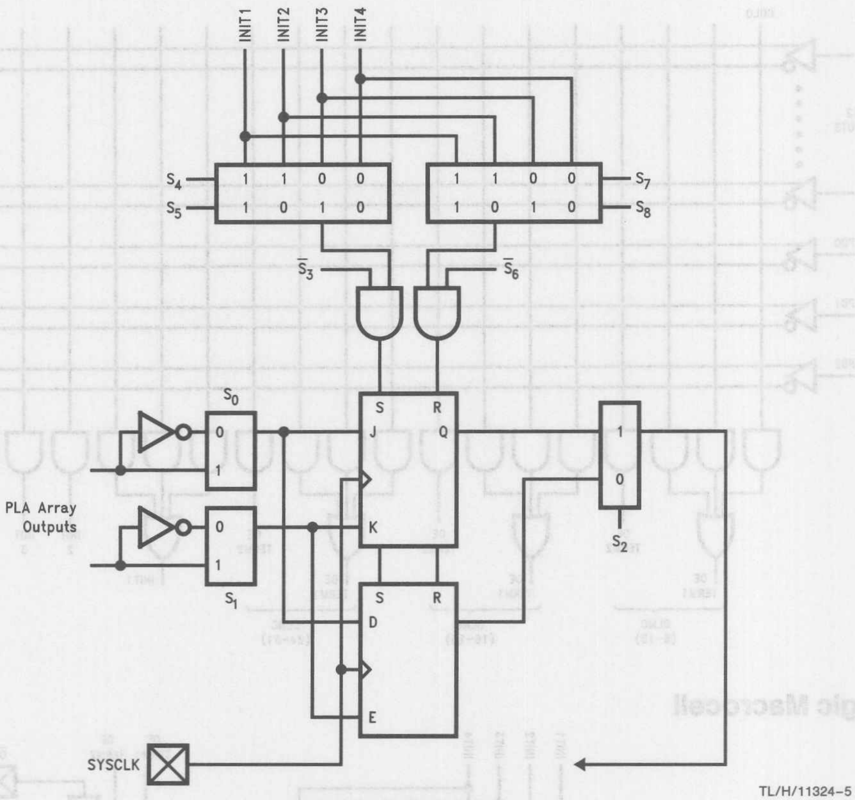
Logic Macrocell



TL/H/11324-3

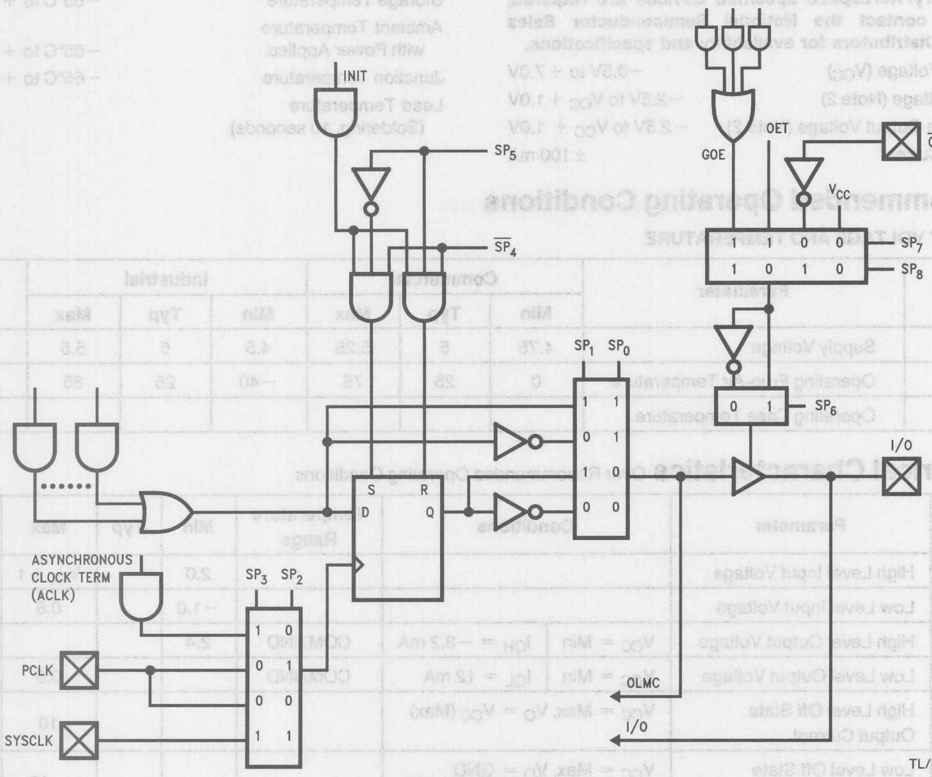
FPLA OLMC (8–31)

Logic Macrocell (Continued)



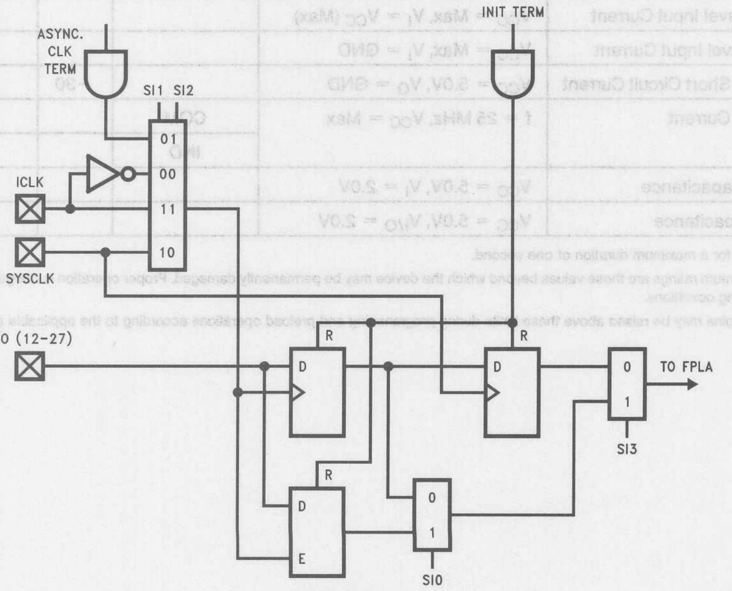
Page/Buried Macrocell

Logic Macrocell (Continued)



TL/H/11324–6

PAL OLMC (0–7)



ILMC Diagram

TL/H/11324–7

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC})	−0.5V to +7.0V
Input Voltage (Note 2)	−2.5V to V_{CC} +1.0V
Off-State Output Voltage (Note 2)	−2.5V to V_{CC} + 1.0V
Output Current	±100 mA
Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−65°C to +125°C
Junction Temperature	−65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	260°C

Recommended Operating Conditions

SUPPLY VOLTAGE AND TEMPERATURE

Symbol	Parameter	Commercial			Industrial			Units
		Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating Free-Air Temperature	0	25	75	−40	25	85	°C
T_C	Operating Case Temperature							°C

Electrical Characteristics Over Recommended Operating Conditions

Symbol	Parameter	Conditions		Temperature Range	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage				2.0		V_{CC}+1	V
V_{IL}	Low Level Input Voltage				−1.0		0.8	V
V_{OH}	High Level Output Voltage	V_{CC} = Min	I_{OH} = −3.2 mA	COM/IND	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min	I_{OL} = 12 mA	COM/IND			0.5	V
I_{OZH}	High Level Off State Output Current	V_{CC} = Max, V_O = V_{CC} (Max)					10	μA
I_{OZL}	Low Level Off State Output Current	V_{CC} = Max, V_O = GND					−10	μA
I_I	Maximum Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = V_{CC} (Max)					10	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = GND					−10	μA
I_{OS}*	Output Short Circuit Current	V_{CC} = 5.0V, V_O = GND			−30		−160	mA
I_{CC}	Supply Current	f = 25 MHz, V_{CC} = Max		COM			180	mA
				IND			200	mA
C_I	Input Capacitance	V_{CC} = 5.0V, V_I = 2.0V					10	pF
$C_{I/O}$	I/O Capacitance	V_{CC} = 5.0V, $V_{I/O}$ = 2.0V					12	pF

*One output at a time for a maximum duration of one second.

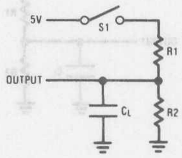
Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. Proper operation is not guaranteed outside specified recommended operating conditions.

Note 2: Some device pins may be raised above these limits during programming and preload operations according to the applicable specification.

Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	
t_{SU}	Setup Time (I/O Pin or Feedback before Clock)	with Speed Option	12		14		17		ns
		without Speed Option	15		17		20		
t_{ISU}	ILMC Setup Time (I/O or OLMC before Clock		5		6		8		ns
t_{CF}	Clock Input to Feedback	S1 Closed $C_L = 35$ pF, with Speed Option		8		11		13	ns
		without Speed Option		5		8		10	
t_H	Hold Time (Input after Clock)		0		0		0		ns
t_W	Clock Pulse Width (High/Low)		6		6		8		ns
t_{CYCLE}	Clock Cycle Period (with OLMC Feedback)		20		25		30		ns
f_{MB}	Clock Frequency with OLMC Feedback*			50		40		33.3	MHz
f_{MO}	Clock Frequency with I/O Feedback*			45.5		40		33.3	MHz
f_{MT}	Clock Frequency without Feedback			83.3		83.3		62.5	MHz
t_{PR}	Clock Valid after Power-Up			100		100		100	ns
t_{CLK}	Clock Input to I/O Pin	S1 Closed, $C_L = 35$ pF, with Speed Option		10		11		13	ns
		without Speed Option		7		8		10	
t_{PZXG}	$\overline{OE}$ ↓ to Registered Output Enabled	Active High; S1 Open, $C_L = 35$ pF Active Low; S1 Closed, $C_L = 35$ pF		10		10		12	ns
t_{PXZG}	$\overline{OE}$ ↑ to Registered Output Disabled	From V_{OL}; S1 Open, $C_L = 5$ pF From V_{OH}; S1 Closed, $C_L = 5$ pF		10		10		12	ns
t_{PZXI}	Input to Output Enable via Product Term	Active High; S1 Open, $C_L = 35$ pF Active Low; S1 Closed, $C_L = 35$ pF		15		15		18	ns
t_{PXZI}	Input to Output Disabled via Product Term	From V_{OL}; S1 Open, $C_L = 5$ pF From V_{OH}; S1 Closed, $C_L = 5$ pF		15		15		18	ns
t_{RESET}	Power Up to Registered Output Low	S1 Closed, $C_L = 35$ pF		45		45		45	µs
t_{IPD}	Input Initialization to Output Reset			15		18		20	ns
t_{IRC}	Input Initialization before Clock			10		12		15	ns

*Actual measurements

AC Test Load



R1 = 400Ω
R2 = 400Ω

TL/H/11324-8

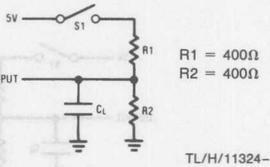
Switching Characteristics Over Recommended Operating Conditions (Continued)

AC TIMING REQUIREMENTS (PAL ARRAY) (Continued)

Symbol	Parameter		Conditions	MAPL244-50		MAPL244-40		MAPL244-33		Units
				COM		COM		COM		
				Min	Max	Min	Max	Min	Max	
t_{SUP}	Setup Time (Input or Feedback before Clock)			10		12		15		ns
t_{HP}	Hold Time (Input after Clock)			0		0		0		ns
t_{PDP}	I/O or OLMC to Combinatorial Output		C_L = 35 pF S1 Closed		15		17		20	ns
t_{CFP}	Clock Input to Feedback				5		8		10	ns
t_{WP}	Clock Pulse Width (High/Low)			6		6		8		ns
t_{CYCLEP}	Clock Cycle Period (with Buried Feedback)			15		20		25		ns
t_{MIP}	Input Frequency				66.6		58.8		50	MHz
f_{MBP}	Clock Frequency	with OLMC Feedback*			66.6		50		40	MHz
f_{MOP}		with I/O Feedback*			52.6		47.6		40	MHz
f_{MTP}		without Feedback			83.3		83.3		62.5	MHz
t_{PRP}	Clock Valid after Power-Up				100		100		100	ns
t_{CLKP}	Clock Input to I/O Pin		S1 Closed, C_L = 35 pF		9		9		10	ns
t_{PZXGP}	$\overline{OE}$ ↓ to Registered Output Enabled		Active High; S1 Open, C_L = 35 pF Active Low; S1 Closed, C_L = 35 pF		10		10		12	ns
t_{PXZGP}	$\overline{OE}$ ↑ to Registered Output Disabled		From V_{OL}; S1 Open, C_L = 5 pF From V_{OH}; S1 Closed, C_L = 5 pF		10		10		12	ns
t_{PZXIP}	Input to Output Enable via Product Term		Active High; S1 Open, C_L = 35 pF Active Low; S1 Closed, C_L = 35 pF		15		15		18	ns
t_{PXZIP}	Input to Output Disabled via Product Term		From V_{OL}; S1 Open, C_L = 5 pF From V_{OH}; S1 Closed, C_L = 5 pF		15		15		18	ns
t_{RESETP}	Power Up to Registered Output Low		S1 Closed, C_L = 35 pF		45		45		45	µs
t_{IPDP}	Input Initialization to Output Reset				20		20		20	ns
t_{IRCP}	Input Initialization before Clock				20		20		20	ns

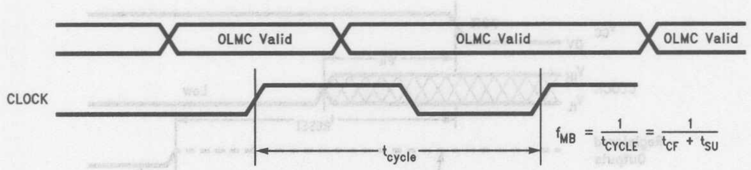
*Actual measurements

AC Test Load



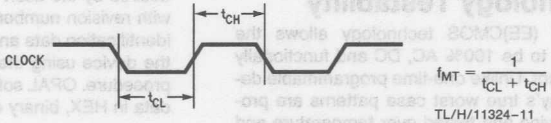
TL/H/11324-9

Timing Diagrams



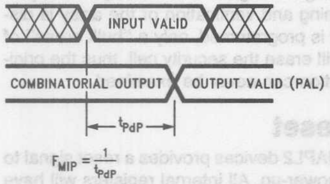
TL/H/11324-10

FIGURE 3. Internal Feedback Frequency



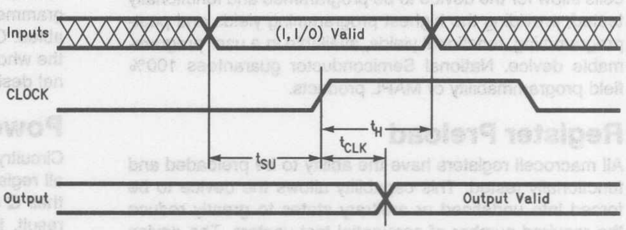
TL/H/11324-11

FIGURE 4. Toggle Frequency



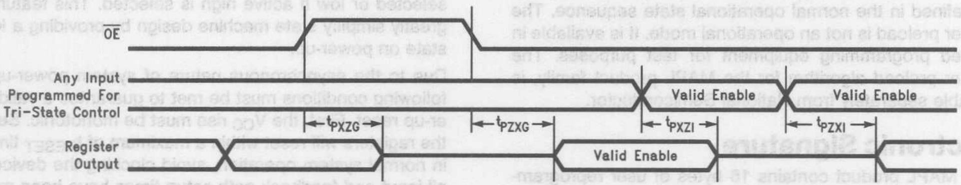
TL/H/11324-12

FIGURE 5. Input Frequency



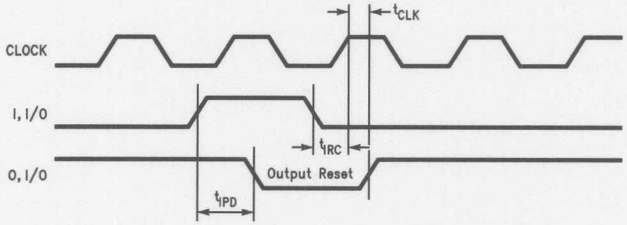
TL/H/11324-13

FIGURE 6. External Feedback



TL/H/11324-14

FIGURE 7. Output Enable Timing



TL/H/11324-15

FIGURE 8. Initialization Timing

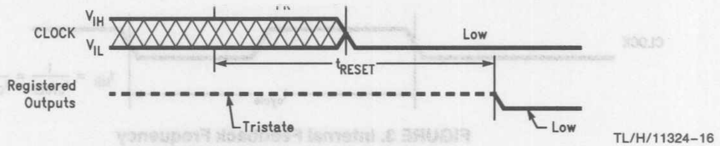


FIGURE 9. Power-Up Reset Timing

Design Technology Testability

Electrically Erasable (EE)CMOS technology allows the MAPL product family to be 100% AC, DC and functionally tested prior to shipment. Unlike one-time programmable devices, the actual array's true worst case patterns are programmed into the device and tested over temperature and voltage ratings. The specifications are guaranteed to be true worst case parameters, with fully loaded internal arrays and all (maximum available) outputs switching. The erasable cells allow for the device to be programmed and functionally tested, providing the highest programming yields and post-programming, functional yields, available in a user programmable device. National Semiconductor guarantees 100% field programmability of MAPL products.

Register Preload

All macrocell registers have the ability to be preloaded and functionally tested. This capability allows the device to be forced into undefined or arbitrary states to greatly reduce the required number of sequential test vectors. The device may be put into any desired register state at any point during the functional test sequence. This allows complete verification of sequential logic circuits including states that are not defined in the normal operational state sequence. The register preload is not an operational mode. It is available in certified programming equipment for test purposes. The register preload algorithm for the MAPL product family, is available separately from National Semiconductor.

Electronic Signature

Each MAPL product contains 16 bytes of user reprogrammable memory known as the User Electronic Signature (UES). The UES may contain any identification information desired by the user. This information typically is associated with revision numbers, dates, inventory control information, identification data and names. The information is read out of the device using the programming equipment's verification procedure. OPAL software allows representation of the UES data in HEX, binary or ASCII formats.

Design Security

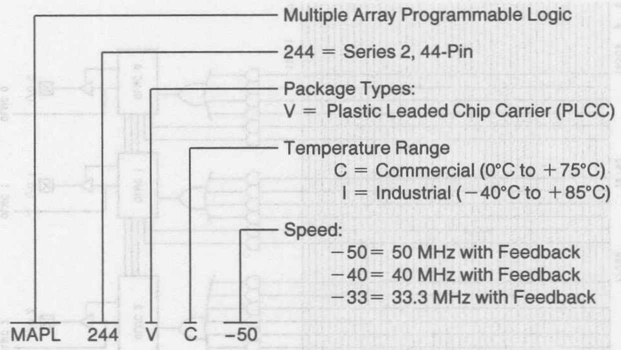
A security cell is provided on all MAPL products as a deterrent to copying the logic design. If the security cell is programmed, programming and verification of the array is disabled. Once the cell is programmed, only a "bulk erase" of the whole product will erase the security cell, thus the original design configuration can never be examined.

Power-Up Reset

Circuitry within the MAPL2 devices provides a reset signal to all registers during power-up. All internal registers will have their Q outputs set low after a specified time (t_{RESET}). As a result, the state on the registered output pins (if they are enabled) will always be low on power-up, except the PAL OLMC's whose polarity control is after the register, in which case the state of the output pins will be high if active low is selected or low if active high is selected. This feature can greatly simplify state machine design by providing a known state on power-up.

Due to the asynchronous nature of system power-up, the following conditions must be met to guarantee a valid power-up reset. First, the V_{CC} rise must be monotonic. Second, the registers will reset within a maximum of t_{RESET} time. As in normal system operation, avoid clocking the device until all input and feedback path setup times have been met.

Ordering Information


Multiple Array Programmable Logic

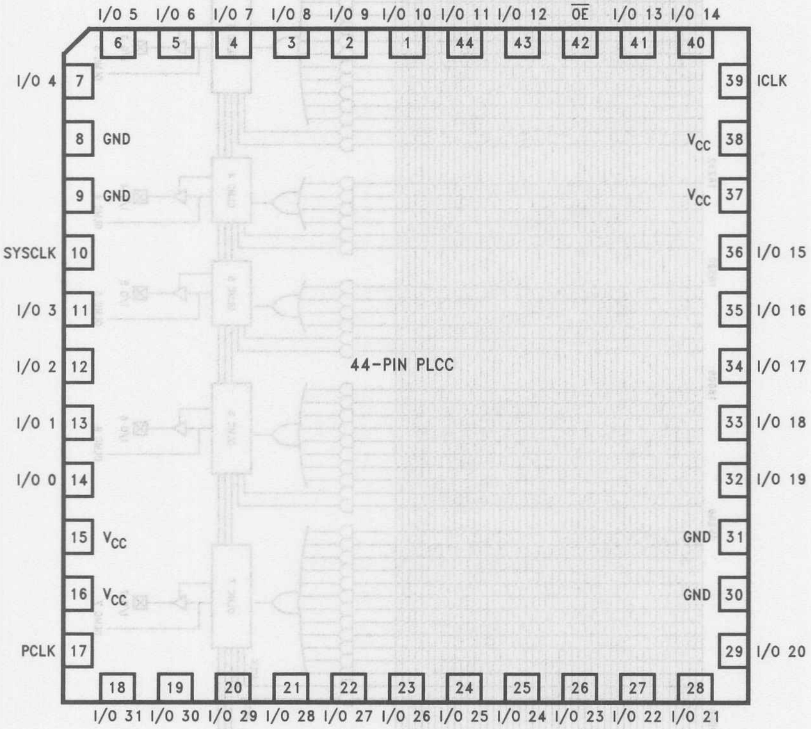
244 = Series 2, 44-Pin

Package Types:
V = Plastic Leaded Chip Carrier (PLCC)

Temperature Range
C = Commercial (0°C to +75°C)
I = Industrial (−40°C to +85°C)

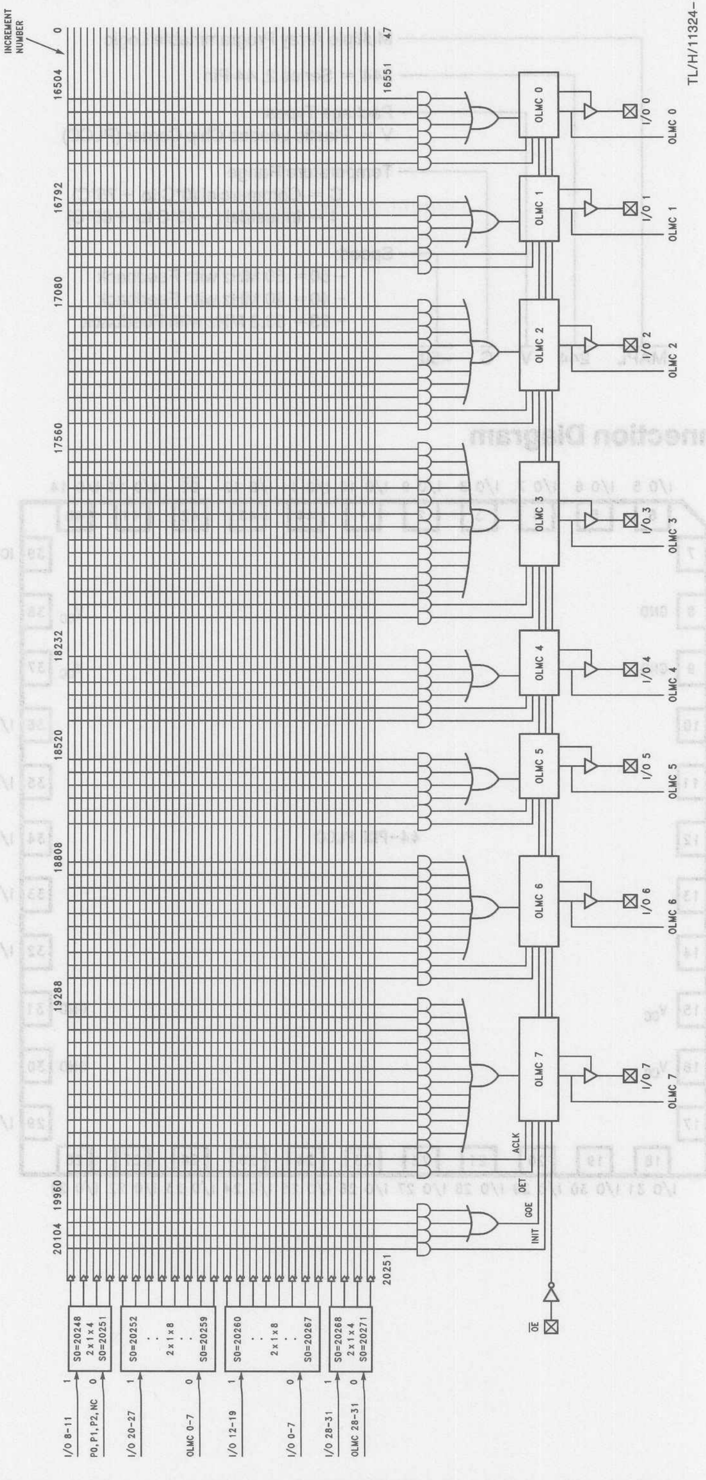
Speed:
−50 = 50 MHz with Feedback
−40 = 40 MHz with Feedback
−33 = 33.3 MHz with Feedback

MAPL244 Connection Diagram



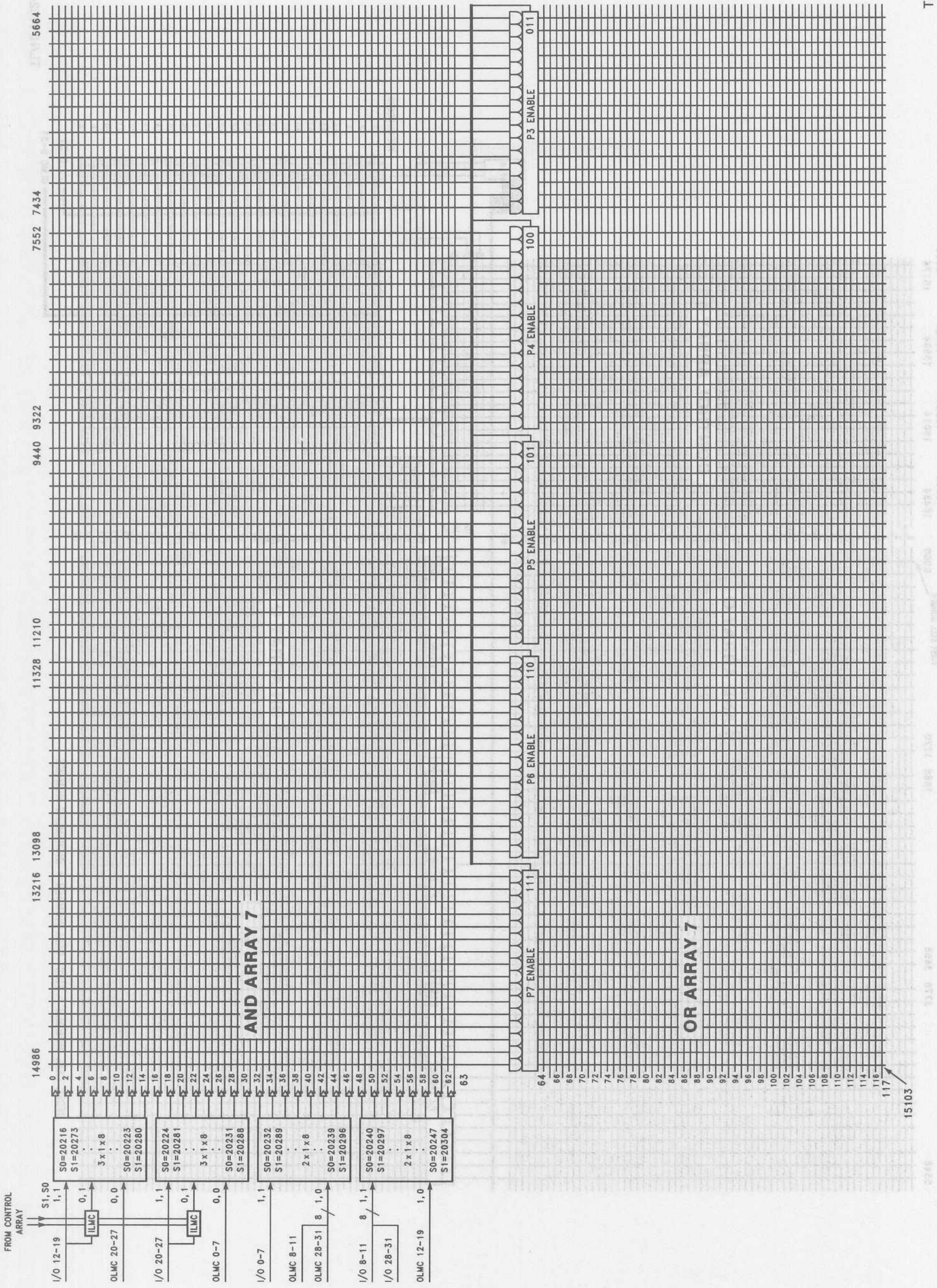
TL/H/11324-17

MAPL244 PAL JEDEC Map



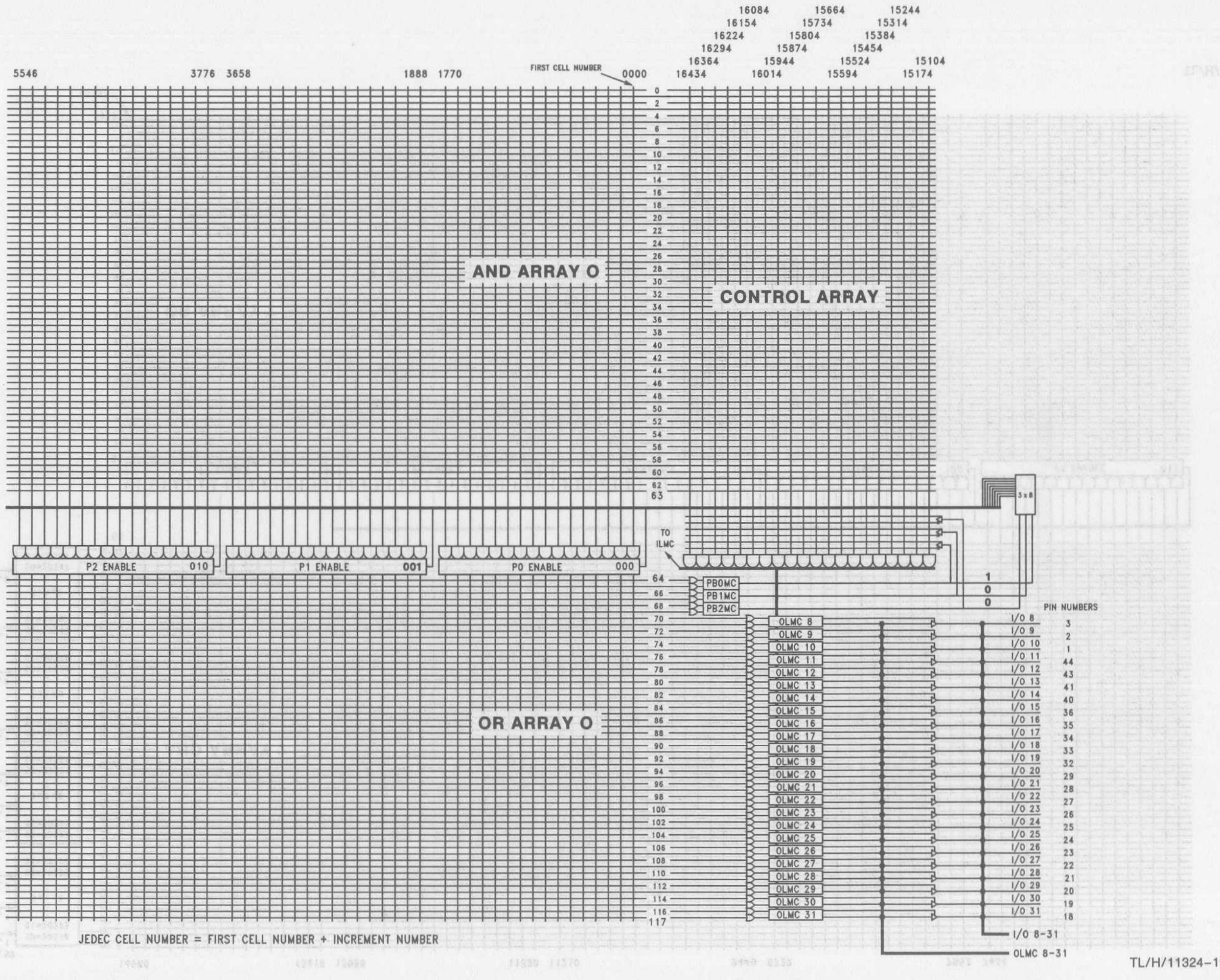
TL/H/11324–18

MAPL244 FPLA JEDEC Map



TL/H/11324-20

MAPL244 FPLA JEDEC Map (Continued)



TL/H/11324-19

MAPL268
Multiple Array Programmable Logic

General Description

The MAPL268 is a second generation device in a series of new higher density, electrically erasable CMOS (EECMOS) programmable logic devices based on a proprietary National Semiconductor programmable logic array architecture. The MAPL268 integrates multiple Field Programmable Logic Arrays (FPLA), allowing for easy implementation of complex state machines, controllers, microinstruction sequencers, bus interfaces and general synchronous logic designs. Additionally the MAPL268 incorporates a separate PAL® block with 8 combinatorial/registered I/O and 8 combinatorial/registered outputs to handle asynchronous and decode functions. The MAPL268 allows for easy and cost effective integration of complex subsystems consisting of synchronous and asynchronous elements.

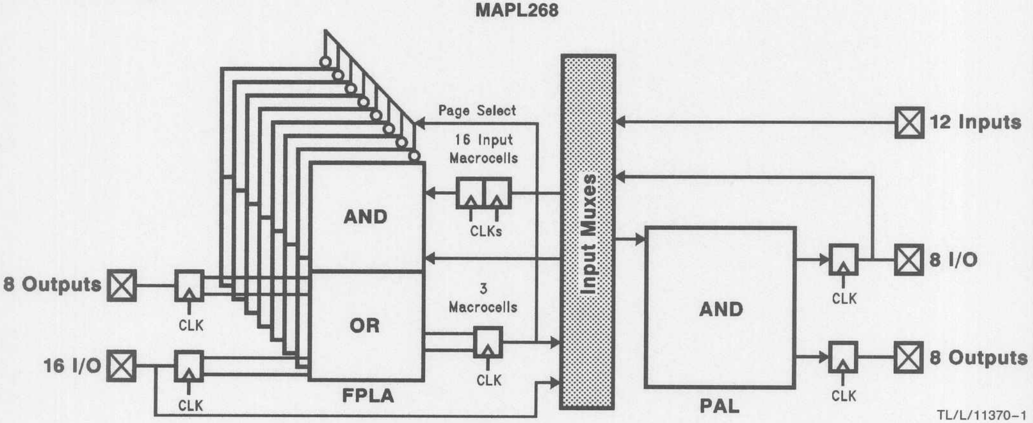
The MAPL™ product family is supported by National Semiconductor's OPAL™ (Open Programmable Architecture Language) software package and by other popular third-party PLD software packages such as ABEL-4, CUPL, PLDesigner and Viewlogic. OPAL is an open environment design tool and is able to interface with these other software packages by means of the de-facto standard Berkeley PLA file format. OPAL also allows the designer to have full access to all of the intermediate files that are generated during the compilation process. Logic partitioning and utilization within a MAPL device may be done 100% automatically or manually using OPAL software. Automatic partitioning and device utilization can also be done using other third-party tools with the appropriate MAPL device fitter, available from the software vendor.

(Continued)

Features

- Extension of MAPL244 architecture
- High density flexible FPLA and PAL architectures
- High performance second generation MAPL architecture combining PAL and FPLA arrays
  - — 15 ns maximum T_{PD}
  - — 50 MHz system performance
- Low power
- 59 total macrocells
  - — 24 I/O macrocells with DE, JK, RS or T-type registers
  - — 16 PAL macrocells (combinatorial or registered)
  - — 3 Buried/Page control macrocells
  - — 16 input macrocells with latched, registered or double registered options
- Multiple clocking and output enable options (synchronous and asynchronous)
- Asynchronous preload, set, reset and power-up reset capability
- Internal and I/O feedback
- Programmable setup and clock-to-output options
- Replaces multiple PAL/GAL/FPLA devices
- 100% functionally tested reconfigurable EECMOS technology
- Fully supported by National's OPAL, and other popular third-party development tools
- Supported by industry standard programmers
- Security cell prevents copying

Block Diagram



General Description (Continued)

Programming MAPL devices is accomplished using readily available, industry standard, PLD programming equipment. NSC guarantees a minimum 100 erase/write cycles for all MAPL devices.

The MAPL268 is available in a 68-pin PLCC package. It conforms to the JEDEC standard for power, ground and clock pin placements.

Product	To/From	Type	MAPL244	MAPL268
I/O Macrocells	FPLA	DE/JK/RS/T	24	16
	PAL	D/Combinatorial	8	8
Dedicated Outputs	FPLA	DE/JK/RS/T	0	8
	PAL	D/Combinatorial	0	8
Buried/Page Macrocells	FPLA	DE/JK/RS/T	3	3
Dedicated Inputs	FPLA/PAL		0	12
Input Logic Macrocells	FPLA	D/L/Double D/Comb.	16	16

National Semiconductor

MAPL™ Application Examples

1.0 MAPL Register Control

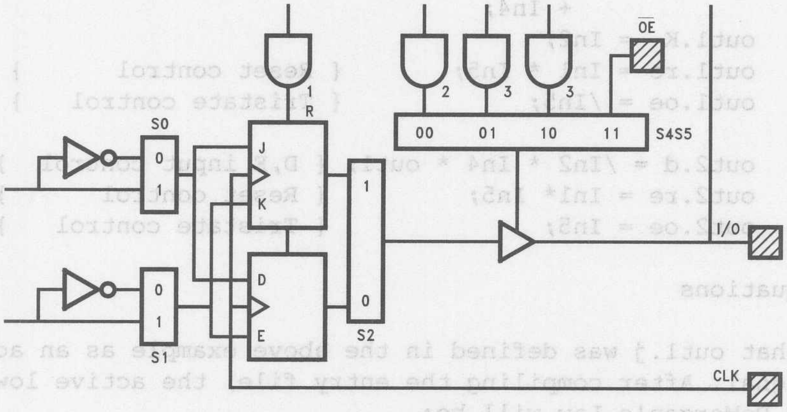
The following application demonstrates the use of OPAL™ to control any single line in the output macrocell of the MAPL128 and configure the output as either a JK or DE register. The example also shows how to drive the reset and the tri-state functions. From the I/O macrocell shown in *Figure 1*, the following signals can be controlled through the OPAL software:

- J input or D input
- K input or E input
- Reset input
- Output Enable

This example uses the OPAL modules individually to generate the JEDEC map for programming the MAPL device:

OPL2PLA CONTROL " will generate PLA file
FITMAPL CONTROL " will generate a MAPL PLA file
PLA2EQN CONTROL " will generate equation file
EQN2JED CONTROL " will generate JEDEC file

The example shows how easy it is to access any of the MAPL OLMC lines and goes step by step using OPAL software to generate desired function.



TL/L/11304–1

FIGURE 1. Input/Output Macrocell

The equations to implement the desired functions are defined as follows:

```
Begin Header
   Title     MAPL I/O control signal
   Pattern   control
   Revision A
   Author    Tarif Arabi
   Date      10/15/90
   End Header

Begin Definition
   Inputs  In1, In2, In3, In4, In5;
   feedbacks (JK)  out1;
   outputs (DE)    out2;
End Definition

Begin Equations

        /out1.j = In1 * /In3       { J,K input control  }
                   + In4;
       out1.K  = In2;
       out1.re = In1 * In5;        { Reset control       }
       out1.oe = /In5;             { Tristate control    }

       out2.d = /In2 * In4 * out1; { D,E input control   }
       out2.re = In1* In5;         { Reset control       }
       out2.oe = In5;              { Tristate control    }

End Equations
```

Note that out1.j was defined in the above example as an active low signal. After compiling the entry file, the active low applying DeMorgan's Law will be:

```
          /out1.j = In1 * /In3 + In4
   will be equivalent to:
          out1.j =  /In1 * /In4 + In3 * /In4
```

Running "OPL2PLA CONTROL" will take CONTROL.OPL as the entry file and generate a PLA file called control.PLA :

```
   #$ TOOL NSC opl2pla B.04
   #$ TITLE          MAPL control signal
   #$ TITLE
   #$ PINS 7 In1 In2 In3 In4 In5 out1 out2
   #$ NODES 0
```

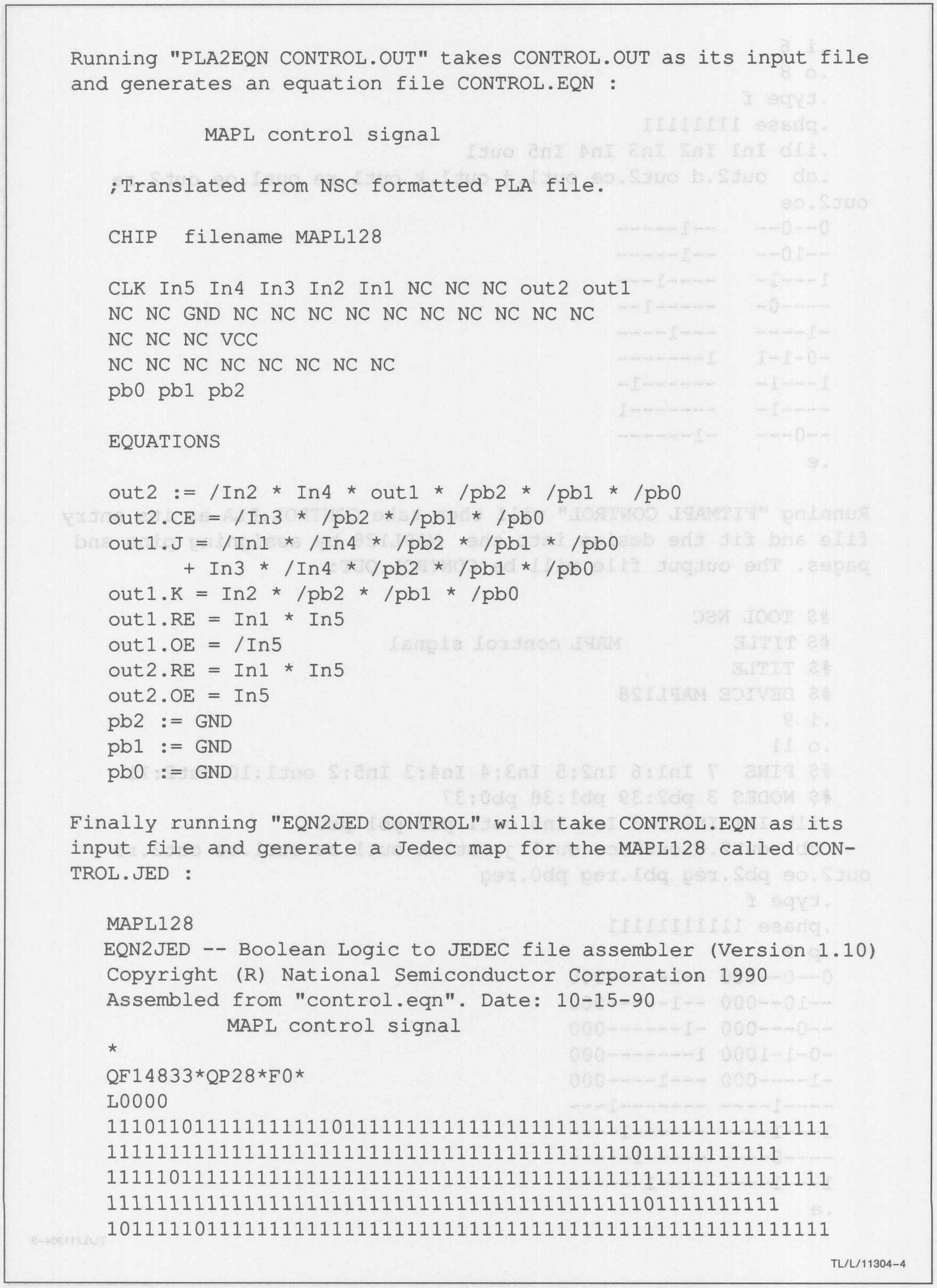
TL/L/11304–2

```
    .i 6
    .o 8
    .type f
    .phase 11111111
    .ilb In1 In2 In3 In4 In5 out1
    .ob  out2.d out2.ce out1.j out1.k out1.re out1.oe out2.re
out2.oe
    0--0--   --1-----
    --10--   --1-----
    1---1-   ----1---
    ----0-   -----1--
    -1----   ---1----
    -0-1-1   1-------
    1---1-   ------1-
    ----1-   -------1
    --0---   -1------
    .e
```

Running "FITMAPL CONTROL" will then take CONTROL.PLA as its entry file and fit the design into the MAPL128 by assigning pins and pages. The output file will be CONTROL.OUT:

```
    #$ TOOL NSC
    #$ TITLE         MAPL control signal
    #$ TITLE
    #$ DEVICE MAPL128
    .i 9
    .o 11
    #$ PINS  7 In1:6 In2:5 In3:4 In4:3 In5:2 out1:10 out2:11
    #$ NODES 3 pb2:39 pb1:38 pb0:37
    .ilb In1 In2 In3 In4 In5 out1 pb2 pb1 pb0
    .ob  out2.d out2.ce out1.j out1.k out1.re out1.oe out2.re
out2.oe pb2.reg pb1.reg pb0.reg
    .type f
    .phase 11111111111
    .p 9
    0--0--000 --1-----000
    --10--000 --1-----000
    --0---000 -1------000
    -0-1-1000 1-------000
    -1----000 ---1----000
    ----1---- -------1~~~
    1---1---- ------1-~~~
    ----0---- -----1--~~~
    1---1---- ----1---~~~
    .e
```

TL/L/11304–3



Running "PLA2EQN CONTROL.OUT" takes CONTROL.OUT as its input file and generates an equation file CONTROL.EQN :

```
          MAPL control signal

   ;Translated from NSC formatted PLA file.

   CHIP  filename MAPL128

   CLK In5 In4 In3 In2 In1 NC NC NC out2 out1
   NC NC GND NC NC NC NC NC NC NC NC NC NC
   NC NC NC VCC
   NC NC NC NC NC NC NC NC
   pb0 pb1 pb2

   EQUATIONS

   out2 := /In2 * In4 * out1 * /pb2 * /pb1 * /pb0
   out2.CE = /In3 * /pb2 * /pb1 * /pb0
   out1.J = /In1 * /In4 * /pb2 * /pb1 * /pb0
         + In3 * /In4 * /pb2 * /pb1 * /pb0
   out1.K = In2 * /pb2 * /pb1 * /pb0
   out1.RE = In1 * In5
   out1.OE = /In5
   out2.RE = In1 * In5
   out2.OE = In5
   pb2 := GND
   pb1 := GND
   pb0 := GND
```

Finally running "EQN2JED CONTROL" will take CONTROL.EQN as its input file and generate a Jedec map for the MAPL128 called CONTROL.JED :

```
   MAPL128
   EQN2JED -- Boolean Logic to JEDEC file assembler (Version 1.10)
   Copyright (R) National Semiconductor Corporation 1990
   Assembled from "control.eqn". Date: 10-15-90
            MAPL control signal
   *
   QF14833*QP28*F0*
   L0000
   1110110111111111110111111111111111111111111111111111111111
   1111111111111111111111111111111111111111110111111111111
   1111101111111111111111111111111111111111111111111111111111
   1111111111111111111111111111111111111111111011111111111
   1011111011111111111111111111111111111111111111111111111111
```

TL/L/11304–4

```
111111111111111111111111111111111111111111110111111111
1111011011111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111110111111111
1101111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111110111111111*
L14336
0111111101111111111111111111111111111111111111111111111111*
L14394
1111111110111111111111111111111111111111111111111111111111*
L14452
1111111011111111111111111111111111111111111111111111111111*
L14510
0000000000000000000000000000000000000000000000000000000000*
L14568
111111111111111111111111111*
L14595
000111111111111111111111111*
L14622
000111111111111111111011111*
L14649
111111111111111111111111*
L14673
1111111111001111*
L14689
1111111111101111*
L14705
0000000000000000000000000000000000000000000000000000000000000000*
L14769
0000000000000000000000000000000000000000000000000000000000000000*
C67DF*
?0000
```

TL/L/11304–5

2.0 A 4-Bit Counter with Reset

Using OPAL, a MAPL128 can be implemented as a 4-bit counter with reset in three different ways. This application demonstrates different ways to enter the same design.

Figure 2 is the entry file for 4-bit counter with reset implemented in a MAPL128 using OPAL truth table entry only.

Figure 3 is the same design implemented in a MAPL128 using OPAL Boolean equation entry only.

Figure 4 is the same design implemented in a MAPL128 using OPAL state diagram entry only.

Note that any of the above entry files will give the same results. To get the JEDEC file run OPL2PLA, FITMAPL, PLA2EQN, and EQN2JED.

```
Begin Header
        Title       4 bit counter
        Pattern     4count
        Revision    A
        Author      Tarif Arabi
        Date        10/15/90
        Everything in the header block is copied directly into
        all the files that generated by OPAL as a comment field
End Header

BEGIN DEFINITION
{ Any thing surrounded by curly brackets is considered to be a
        comment}

Device MAPl128;

INPUTS
    RESET;

feedbacks (JK,HOLD)                        {Define 4 I/O as a JK}
                                           {default to hold}
    CNT4,CNT3,CNT2,CNT1;
END DEFINITION

BEGIN truth_TABLE
ttin     RESET,CNT4,CNT3,CNT2,CNT1;
ttout    CNT4,CNT3,CNT2,CNT1;

1 ---- 0000
0 ---- ---!
0 ---1 --!-
0 --11 -!--
0 -111 !---

END truth_TABLE
```

TL/L/11304-6

FIGURE 2. OPAL Truth Table

```
begin header
   4 bit counter design using state diagram entry format only
end header

begin definition
inputs
   reset;
statebits
   cnt4,cnt3,cnt2,cnt1;
end definition

begin state_diagram

state zero : if reset !$ 0 then one else zero;
state one : if reset !$ 0 then two else zero;
state two : if reset 1$ 0 then three else zero;
state three : if reset !$ 0 then four else zero;
state four : if reset !$ 0 then five else zero;
state five : if reset !$ 0 then six else zero;
state six : if reset !$ 0 then seven else zero;
state eight : if reset !$ 0 then nine else zero;
state nine : if reset !$ 0 then ten else zero;
state ten : if reset !$ 0 then eleven else zero;
state eleven : if reset !$ 0 then twelve else zero;
state twelve : if reset !$ 0 then thirteen else zero;
state thirteen : if reset !$ 0 then fourteen else zero;
state fourteen : if reset !$ 0 then fifteen else zero;
state fifteen : goto zero;

end state_diagram
```

TL/L/11304-7

FIGURE 3. State Diagram Entry Format

```
begin header
   4 bit counter design using boolean equations only
end header

begin Definition

inputs reset;
feedbacks    cnt4,cnt3,cnt2,cnt1;

end Definition

begin Equations

cnt4  := /reset * /cnt4 * cnt3 * cnt2 * cntl
       + /reset * cnt4 * /cnt1
       + /reset * cnt4 * /cnt3
       _ /reset * cnt4 * /cnt2;
cnt3  := /reset * /cnt3 * cnt2 * cnt1
       _ /reset * cnt3 * /cnt2
       + /reset * cnt3 * /cnt1;
cnt2  := /reset * /cnt2 * cnt1
       + /reset * cnt2 * /cnt1;
cnt1  := /reset * /cnt1;

end Equations
```

TL/L/11304-8

FIGURE 4. Boolean Equations

3.0 A 3-Bit Up-Down Counter with 7-Segment Display Output

This application demonstrates the use of OPAL's state diagram entry format and set assignment to implement this design. The counter will reset to zero if rst is low, it will count:

- upward if up is high,
- downward if down is high,
- upward if both are high,
- hold value if both are low.

This application shows how to implement the above mentioned design using state diagram entry. This design can also be entered using the truth table format.

3_to_7.OPL

```
begin header
           Company    :  National Semiconductor Corp.
           Author     :  Tarif Arabi
           Date       :  11/15/91
           Revision   :  B
           Description:  A three-bit up-down counter with 7-segment display
                         output.

end header

begin definition
           device mapl128;
           inputs up, down, rst;

        { 7-segment display output}
           outputs  a, b, c, d, e, f, g;

        { symbols to define state names and their values, statebits
          will be assigned automatically}
           symbols zero=0, one=^b1, two=^b10, three=3, four=4,
                   five=5, six=6, seven=7;

        { Sets are defined to group inputs and outputs}
           Sets out_disp= [a,b,c,d,e,f,g], ZERO= [1,1,1,1,1,1,0],
                TWO  = [1,1,0,1,1,0,1], THREE= [1,1,1,1,0,0,1],
                FOUR = [0,1,1,0,0,1,1], FIVE = [1,0,1,1,0,1,1],
                SIX  = [1,0,1,1,1,1,1], SEVEN= [1,1,1,0,0,0,0];
end definition
```

TL/L/11304-9

```
 begin state_diagram
         state zero :
            out_disp = ZERO;             { out '0' to 7-segment display}
            if /rst then zero            { reset counter }
            else if up then one          { count upward }
            else if down then seven      { count downward }
            else zero;                   { hold }

         state one :
            out_disp = [0,1,1,0,0,0,0];  { set notation was used rather than
                                           a defined set  to display '1'}

            if /rst then zero
            else if up then two
            else if down then zero
            else one;

         state two :
            out_disp = TWO;
            if /rst then zero
            else if up then three
            else if down then one
            else two;

         state three :
            out_disp = THREE;
            if /rst then zero
            else if up then four
            else if down then two
            else three;

           else if down then three
           else four;

        state five :
           out_disp = FIVE;
           if /rst then zero
           else if up then six
           else if down then four
           else five;

        state six :
           out_disp = SIX;
           if /rst then zero
           else if up then seven
           else if down then five
           else six;

        state seven :
           out_disp = SEVEN;
           if /rst then zero
           else if up then zero
           else if down then six
           else seven;

end state_diagram
```

TL/L/11304-10

4.0 15-Bit Up/Down Counter

This application demonstrates the use of the MAPL128 in a state machine design to implement a 15-bit up/down counter with reset and hold. The JEDEC map for the MAPL128 was generated using OPAL software.

This manual paging example shows how a designer can use OPAL to generate JEDEC maps for the MAPL products by completely specifying all pins and page data thereby eliminating the need to execute FITMAPL.

The design was simulated using the Verilog simulator. This application shows the implementation of the design in a truth table, while if the same design was entered using the state diagram entry format, 32,768 states would appear in the entry file.

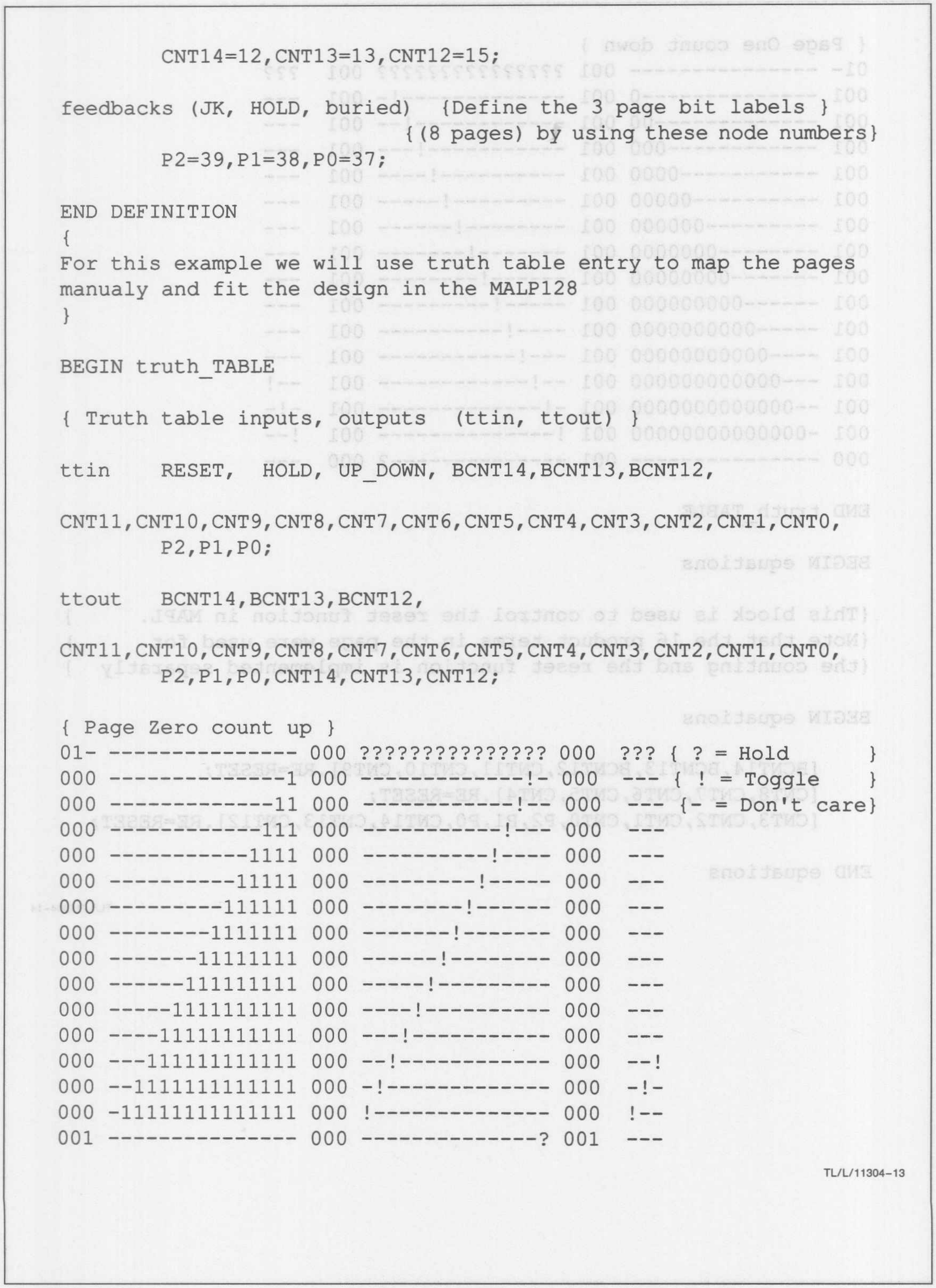
To implement the same design in a regular PAL or GAL device would take roughly four PAL16R4's. In this application, 25% of the MAPL128 was used to implement the 15-bit counter (2 pages out of 8).

To get the JEDEC file run the following modules:

- OPL2PLA 15UPDN
- PLA2EQN 15UPDN
- EQN2JED 15UPDN

```
Begin Header
    Title     15 bit up/down counter
    Pattern   15UPDN
    Revision  D
    Author    Tarif Arabi
    Date      12/10/90

    Everything in the header command is copied directly into the
    jedec map as a comment field
End Header

BEGIN DEFINITION
{
Any thing surrounded by curly brackets is considered to be a com-
ment
}

Device MAPL128;                        { Specify the device used    }

INPUTS                                 { Define 3 inputs            }

         RESET=6,  HOLD=5, UP_DOWN=4;

feedbacks (JK, HOLD, buried)  {Define 3 buried register as count}
                             {bits configured as JK with Hold default}

         BCNT14=29,BCNT13=30,BCNT12=31;

feedbacks (JK,HOLD)                    {Define 11 I/O as a JK       }
                                       {default to hold             }


CNT11=11,CNT10=10,CNT9=9,CNT8=8,CNT7=7,CNT6=17,CNT5=18,CNT4=19,
          CNT3=20,CNT2=21,CNT1=22;

feedbacks (JK,TOGGLE)                  {Define one I/O as JK toggling}
                                       {always with the clock       }
CNT0=23;

OUTPUTS (JK, HOLD)      {Define 3 outputs as JK so the total number
}
                       {of outputs will be 15.  Observe that in the  }
                       {truth table, they are driven by the buried   }
                      {feedback, which actually represents the count}
```

TL/L/11304-12

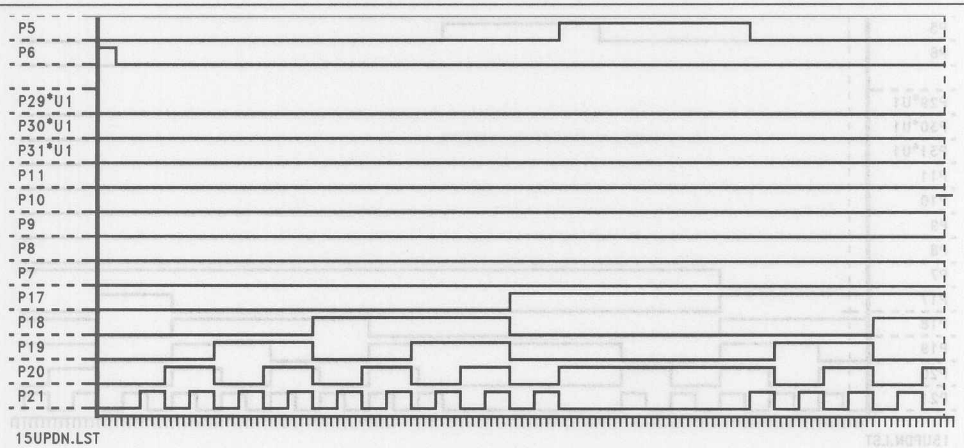


```
        CNT14=12,CNT13=13,CNT12=15;

feedbacks (JK, HOLD, buried)  {Define the 3 page bit labels }
                            {(8 pages) by using these node numbers}
        P2=39,P1=38,P0=37;

END DEFINITION
{
For this example we will use truth table entry to map the pages
manualy and fit the design in the MALP128
}

BEGIN truth_TABLE

{ Truth table inputs, outputs  (ttin, ttout) }

ttin    RESET,  HOLD, UP_DOWN, BCNT14,BCNT13,BCNT12,

CNT11,CNT10,CNT9,CNT8,CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0,
        P2,P1,P0;

ttout   BCNT14,BCNT13,BCNT12,

CNT11,CNT10,CNT9,CNT8,CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0,
        P2,P1,P0,CNT14,CNT13,CNT12;

{ Page Zero count up }
01- --------------- 000 ??????????????? 000  ??? { ? = Hold      }
000 --------------1 000 --------------! 000  --- { ! = Toggle    }
000 -------------11 000 -------------!- 000  --- {- = Don't care}
000 ------------111 000 ------------!-- 000  ---
000 -----------1111 000 -----------!--- 000  ---
000 ----------11111 000 ----------!---- 000  ---
000 ---------111111 000 ---------!----- 000  ---
000 --------1111111 000 --------!------ 000  ---
000 -------11111111 000 -------!------- 000  ---
000 ------111111111 000 ------!-------- 000  ---
000 -----1111111111 000 -----!--------- 000  ---
000 ----11111111111 000 ----!---------- 000  ---
000 ---111111111111 000 ---!----------- 000  --!
000 --1111111111111 000 --!------------ 000  -!-
000 -11111111111111 000 -!------------- 000  !--
001 --------------- 000 --------------? 001  ---
```

TL/L/11304-13

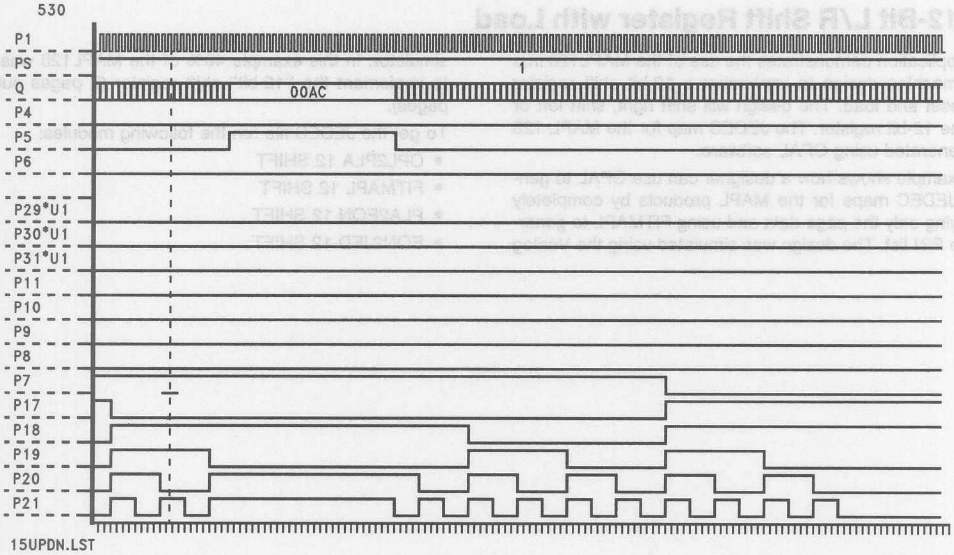
```
{ Page One count down }
01- --------------- 001 ??????????????? 001  ???
001 --------------0 001 -------------!- 001  ---
001 -------------00 001 ------------!-- 001  ---
001 ------------000 001 -----------!--- 001  ---
001 -----------0000 001 ----------!---- 001  ---
001 ----------00000 001 ---------!----- 001  ---
001 ---------000000 001 --------!------ 001  ---
001 --------0000000 001 -------!------- 001  ---
001 -------00000000 001 ------!-------- 001  ---
001 ------000000000 001 -----!--------- 001  ---
001 -----0000000000 001 ----!---------- 001  ---
001 ----00000000000 001 ---!----------- 001  ---
001 ---000000000000 001 --!------------ 001  --!
001 --0000000000000 001 -!------------- 001  -!-
001 -00000000000000 001 !-------------- 001  !--
000 --------------- 001 --------------? 000  ---

END truth_TABLE

BEGIN equations

{This block is used to control the reset function in MAPL.     }
{Note that the 16 product terms in the page were used for      }
{the counting and the reset function is implemented separatly  }

BEGIN equations

    [BCNT14,BCNT13,BCNT12,CNT11,CNT10,CNT9].RE=RESET;
    [CNT8,CNT7,CNT6,CNT5,CNT4].RE=RESET;
    [CNT3,CNT2,CNT1,CNT0,P2,P1,P0,CNT14,CNT13,CNT12].RE=RESET;

END equations
```

TL/L/11304-14



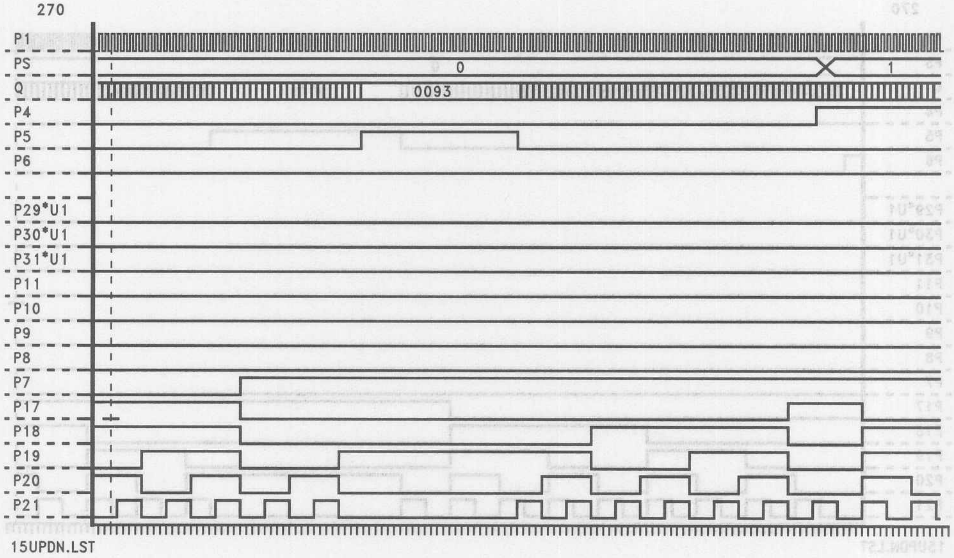
TL/L/11304-15

FIGURE 5. Timing Diagram



TL/L/11304-16

FIGURE 6. Timing Diagram



TL/L/11304–17

FIGURE 7. Timing Diagram

5.0 12-Bit L/R Shift Register with Load

This application demonstrates the use of the MAPL128 in a state machine design to implement a 12-bit shift register with reset and load. The design will shift right, shift left or load the 12-bit register. The JEDEC map for the MAPL 128 was generated using OPAL software.

This example shows how a designer can use OPAL to generate JEDEC maps for the MAPL products by completely specifying only the page data and using FITMAPL to generate the PIN list. The design was simulated using the Verilog simulator. In this example 40% of the MAPL128 was used to implement the "12-bit" shift register (3 pages out of 8 pages).

To get the JEDEC file run the following modules:

- OPL2PLA 12 SHIFT
- FITMAPL 12 SHIFT
- PLA2EQN 12 SHIFT
- EQN2JED 12 SHIFT

```
{
 Entry file for a 12bit shift register - manual paging example.
}
Begin Header

Title     12 bits shift right/left with load
Pattern   12shift
Revision  C
Author    Tarif Arabi
Date      12/10/90

Everything in the header command is copied directly into the
jedec map as a comment field

End Header

BEGIN DEFINITION
{
Any thing surrounded by curly brackets are considered to be a
comment
 }
```

TL/L/11304-18

```
  Device MAPL128;

  INPUTS                       { Define 5 inputs          }

      RESET,  HOLD, SH, LD, SHIN;

  FEEDBACKS (DE,RST)           {Define 12 I/O as DE register }
                               {default to reset             }

          S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0;

  FEEDBACKS (JK, HOLD, BURIED)

      {Define the page labels because manual paging is used}

         p2=39,p1=38,p0=37;

  END DEFINITION


  BEGIN TRUTH_TABLE

  ttin RESET,  HOLD,SH,LD,SHIN,S11,S10,S9,S8,S7,
               S6,S5,S4,S3,S2,S1,S0,p2,p1,p0;

  ttout  S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0,p2,p1,p0;
```

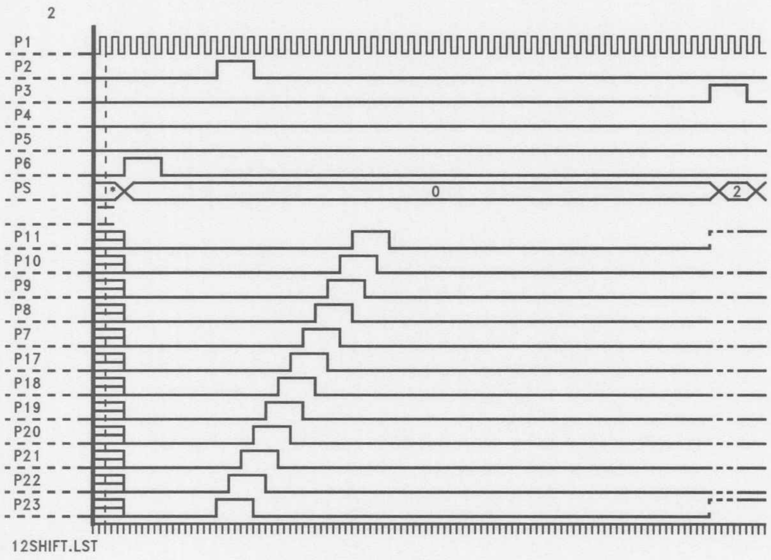
TL/L/11304-19

```
{
Input  I/O          Page   Next I/O  Next
                                     page  }
01000 ------------ 000 ???????????? 000
00001 ------------ 000 -----------1 000   {Shift left when SH = L}
0000- -----------1 000 ----------1- 000
0000- ----------1- 000 ---------1-- 000
0000- ---------1-- 000 --------1--- 000
0000- --------1--- 000 -------1---- 000
0000- -------1---- 000 ------1----- 000
0000- ------1----- 000 -----1------ 000
0000- -----1------ 000 ----1------- 000
0000- ----1------- 000 ---1-------- 000
0000- ---1-------- 000 --1--------- 000
0000- --1--------- 000 -1---------- 000
0000- -1---------- 000 1----------- 000
0010- ------------ 000 ???????????? 001
0001- ------------ 000 ------------ 010

01--- ------------ 001 ???????????? 001
00101 ------------ 001 1----------- 001 {Shift right when SH = H}
0010- ----------1- 001 -----------1 001
0010- ---------1-- 001 ----------1- 001
0010- --------1--- 001 ---------1-- 001
0010- -------1---- 001 --------1--- 001
0010- ------1----- 001 -------1---- 001
0010- -----1------ 001 ------1----- 001
0010- ----1------- 001 -----1------ 001
0010- ---1-------- 001 ----1------- 001
0010- --1--------- 001 ---1-------- 001
0010- -1---------- 001 --1--------- 001
0010- 1----------- 001 -1---------- 001
0000- ------------ 001 ???????????? 000
0001- ------------ 001 ------------ 010
```

TL/L/11304–20

```
01--- ------------ 010 ???????????? 010
0001- -----------1 010 -----------1 010  {Load when LD = H  }
0001- ----------1- 010 ----------1- 010
0001- ---------1-- 010 ---------1-- 010
0001- --------1--- 010 --------1--- 010
0001- -------1---- 010 -------1---- 010
0001- ------1----- 010 ------1----- 010
0001- -----1------ 010 -----1------ 010
0001- ----1------- 010 ----1------- 010
0001- ---1-------- 010 ---1-------- 010
0001- --1--------- 010 --1--------- 010
0001- -1---------- 010 -1---------- 010
0001- 1----------- 010 1----------- 010
0000- ------------ 010 ???????????? 000
0010- ------------ 010 ???????????? 001

END TRUTH_TABLE

   {Disable the I/O when loading (LD = 1)  }
   {I/O used as input to load the new value}

BEGIN equation

 [S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0].oe= /LD;

END equation
```

TL/L/11304-21



TL/L/11304-22

FIGURE 8. Timing Diagram

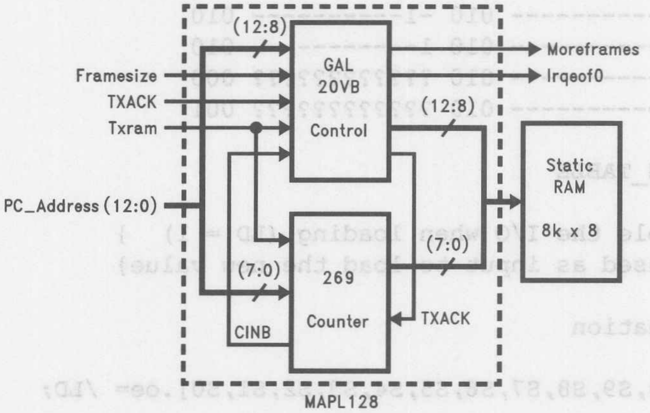
6.0 High Speed Frame Buffer Control

INTRODUCTION

This application demonstrates how the MAPL family can integrate PAL, GAL, and discrete logic devices. Specifically, this application note describes the merging of a GAL20V8 and a counter into a MAPL128 on the FDDI MAC Layer Evaluation Board with a reduction in power and board space. The integration of functions is done using OPAL software along with a multiple level logic partitioning algorithm. It is assumed that the reader is already familiar with the FDDI MAC Layer Evaluation Board and multiple level logic partitioning.

Description of Design

The block diagram in *Figure 9* shows which devices and I/O signals are merged into one MAPL128 device. The GAL portion of the initial design contains the upper five bits of the address which accesses the SRAM, along with some extra decode functions. The input control signals are received from the mode register (FRAMESIZE), the PC interface module (TXRAM), and the BMAC (TXACK). The design outputs a IRQEOF0 signal back to the BMAC, and drives the MOREFRAMES signal for the transmit sequencer. *Figure 10* shows where this block fits into the entire block diagram of the FDDI MAC Layer Evaluation Board.

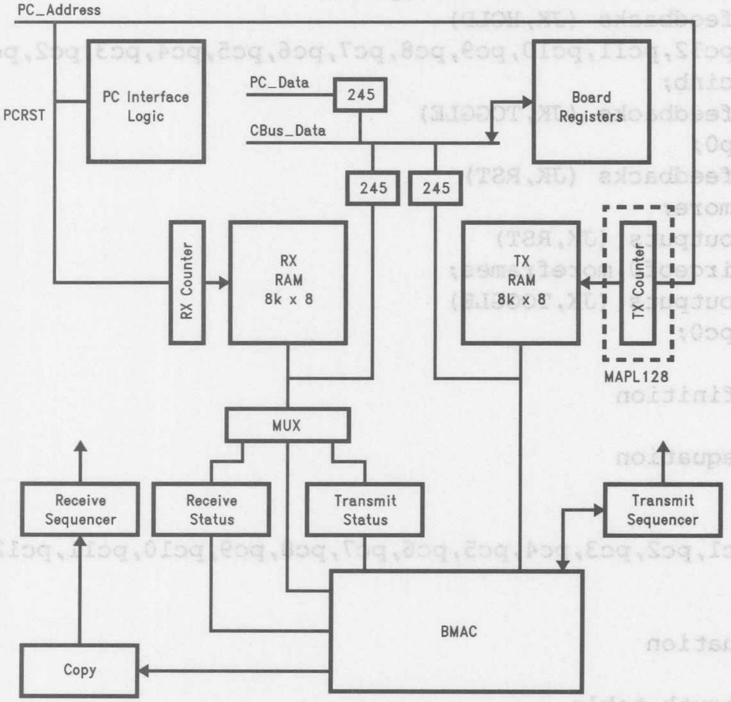


TL/L/11304–23

FIGURE 9. Block Diagram

The MAPL design also cuts down on a few I/O signals between the GAL and the counter. The three I/O signals that are integrated within the MAPL design are CINB, TXACK~, and TXRAM~. The CINB signal is the carry in bit from the 269 which triggers the GAL to count through the high order address bits. The signal TXACK is inverted through the GAL and then fed to the counter. TXRAM~ is used as input to both blocks and thus can be integrated. So, not only can the MAPL reduce board space and power consumption, it can also eliminate some PCB routing.

Figure 11 shows the implementation of the design in OPAL's high end .OPL file format. The .OPL file gives the designer freedom to choose state machine language, multilevel boolean equations, enhanced truth table functions, or any combination of the three to implement a design. For this specific design, it is easier to enter a truth table function for a 13-bit counter instead of a numerous amount of Boolean equations. Notice that in the .OPL file for this design, Boolean equations are used along with the truth table function for output enables. Actually, two functions were integrated into the .OPL file; namely the GAL control logic and the counter.



TL/L/11304-24

FIGURE 10. FDDI MAC Board Block Diagram Data Path

```
begin header
      transmit counter
end header

begin definition

      inputs
      txram,txack,framesize,pcin;
      feedbacks (JK,HOLD)
      pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,pc2,pc1,
      cinb;
      feedbacks (JK,TOGGLE)
      p0;
      feedbacks (JK,RST)
      more;
      outputs (JK,RST)
      irqeof0,moreframes;
      outputs (JK,TOGGLE)
      pc0;

end definition

begin equation


[pc0,pc1,pc2,pc3,pc4,pc5,pc6,pc7,pc8,pc9,pc10,pc11,pc12].oe =
txram;

end equation

begin truth_table

      ttin
txram,txack,framesize,pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,
pc3,pc2,pc1,p0,pcin,more,cinb;
      ttout
pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,pc2,pc1,p0,
pc0,more,moreframes,irqeof0,cinb;
```

TL/L/11304–25

FIGURE 11. Transmit Counter

```
            0-- -------------1-0 ------------11---0
            0-- -----------1---0 -----------1-----0
            0-- ----------1----0 ----------1------0
            0-- ---------1-----0 ---------1-------0
            0-- --------1------0 --------1--------0
            0-- -------1-------0 -------1---------0
            0-- ------1--------0 ------1----------0
            0-- -----1---------0 -----1-----------0
            0-- ----1----------0 ----1------------0
            0-- ---1-----------0 ---1-------------0
            0-- --1------------0 --1--------------0
            0-- -1-------------0 -1---------------0
            0-- 1--------------0 1----------------0
            0-- ---------------0 -----------------0
            11- ------------0--0 -----------!--1110
            11- -----------00--0 ----------!---1110
            11- ----------000--0 ---------!----1110
            11- ---------0000--0 --------!-----1110
            11- --------00000--0 -------!------1110
            11- -------000000--0 ------!-------1110
            11- ------0000000--0 -----!-----------0
            11- -----00000000--0 ----!-----------11
            11- ----000000000--0 ---!------------11
            11- ---0000000000--0 --!-------------11
            11- --00000000000--0 -!--------------11
            11- -000000000000--0 !---------------11
            1-- ----1---------11 --------------11-0
            1-- ---1----------11 --------------11-0
            1-- --1-----------11 --------------11-0
            1-- -1------------11 --------------11-0
            1-- 1-------------11 --------------11-0
            11- --------------01 --------------11-0
            -1- --------------10 --------------11-0
            --1 1--------------1 ----------------10
            --1 -1-------------1 ----------------10
            --1 --1------------1 ----------------10
            --1 ---1-----------1 ----------------10
            --- ----1----------1 ----------------10
            --- ---------------0 ----------------10
            -0- ---------------0 ----------------10

end truth_table
```

TL/L/11304–26

FIGURE 11. Transmit Counter (Continued)

The file in *Figure 11* will not fit into the MAPL device due to partitioning problems. Refer to "Integrating Multiple Functions into a High Density PLD Using Efficient Mapping Algorithms" in the software section. Thus, the multiple level logic partitioning algorithm must be performed to allow the PLA file to be be partitioned. An XLT term is added to the .OPL file which is shown in *Figure 12*. Once the multiple level logic partitioning algorithm is performed, the MAPL fitter program will partition the logic and allocate it to "pages" in the device.

The output from the fitter program can then be used with the OPAL software to obtain an equations file as well as a JEDEC map. Using the JEDEC map and the OPALSIM simulator package, the design can be tested and verified. The OPALSIM output waveforms are illustrated in *Figure 13* through *Figure 15*.

```
begin header
   transmit counter
end header

begin definition

   inputs
      txram,txack,framesize,pcin;
   feedbacks (JK,HOLD)
      pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,pc2,pc1,
      cinb;
   feedbacks (JK, RST)
      pg;
   feedbacks (JK,TOGGLE)
      p0;
   feedbacks (JK,RST)
      more;
   outputs (JK,RST)
      irqeof0,moreframes;
   outputs (JK,TOGGLE)
      pc0;

end definition

begin equation

     [pc0,pc1,pc2,pc3,pc4,pc5,pc6,pc7,pc8,pc9,pc10,pc11,
pc12].oe = txram;

end equation

begin truth_table

   ttin txram,txack,framesize,pc12,pc11,pc10,pc9,pc8,pc7,pc6,
      pc5,pc4,pc3,pc2,pc1,p0,pcin,more,pg,cinb;
   ttout pc12,pc11,pc10,pc9,pc8,pc7,pc6,pc5,pc4,pc3,
      pc2,pc1,p0,pc0,more,moreframes,irqeof0,pg,cinb;
```

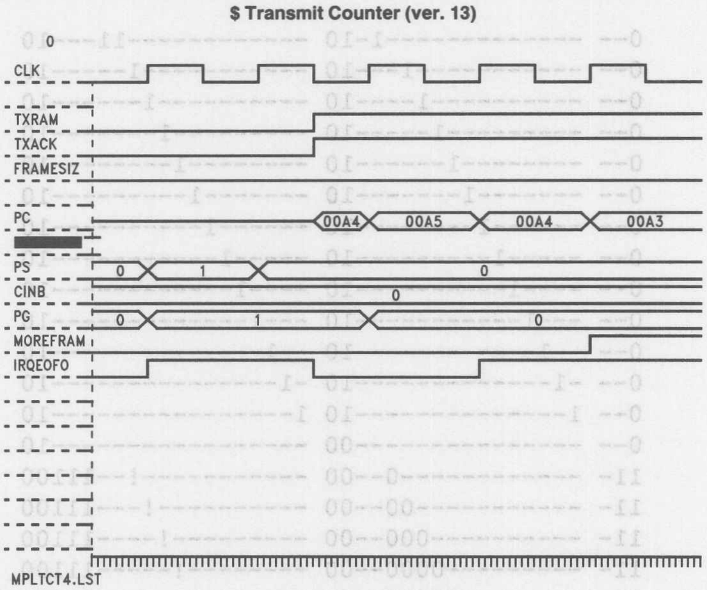
TL/L/11304-27

FIGURE 12. Transmit Counter XLT

```
    0-- -------------1-10 ------------11---10
    0-- -----------1---10 -----------1-----10
    0-- ----------1----10 ----------1------10
    0-- ---------1-----10 ---------1-------10
    0-- --------1------10 --------1--------10
    0-- -------1-------10 -------1---------10
    0-- ------1--------10 ------1----------10
    0-- -----1---------10 -----1-----------10
    0-- ----1----------10 ----1------------10
    0-- ---1-----------10 ---1-------------10
    0-- --1------------10 --1--------------10
    0-- -1-------------10 -1---------------10
    0-- 1--------------10 1----------------10
    0-- ---------------00 -----------------10
    11- ------------0--00 -----------!--11100
    11- -----------00--00 ----------!---11100
    11- ----------000--00 ---------!----11100
    11- ---------0000--00 --------!-----11100
    11- --------00000--00 -------!------11100
    11- -------000000--00 ------!-------11100
    11- ------0000000--00 -----!-----------00
    11- -----00000000--00 ----!-----------101
    11- ----000000000--00 ---!------------101
    11- ---0000000000--00 --!-------------101
    11- --00000000000--00 -!--------------101
    11- -000000000000--00 !---------------101
    1-- ----1---------101 --------------11-00
    1-- ---1----------101 --------------11-00
    1-- --1-----------101 --------------11-00
    1-- -1------------101 --------------11-00
    1-- 1-------------101 --------------11-00
    11- --------------001 --------------11-00
    -1- --------------100 --------------11-00
    --1 1--------------01 ----------------100
    --1 -1-------------01 ----------------100
    --1 --1------------01 ----------------100
    --1 ---1-----------01 ----------------100
    --- ----1----------01 ----------------100
    --- ---------------00 ----------------100
    -0- ---------------00 ----------------100

end truth_table
```

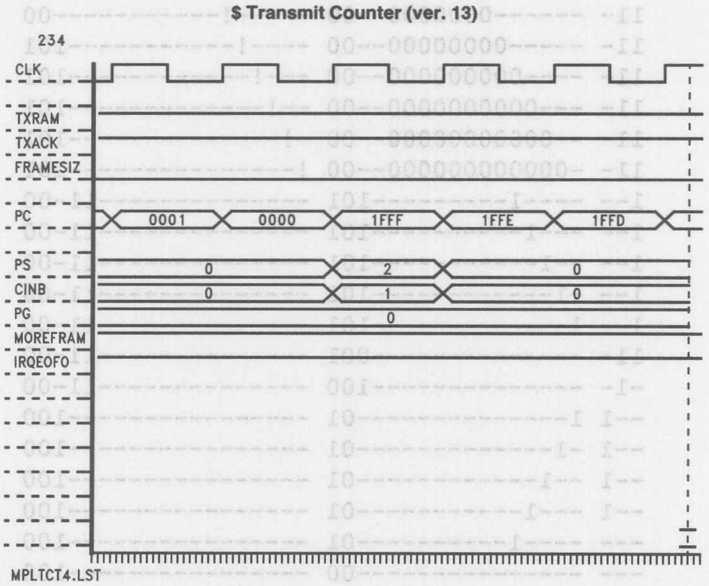
TL/L/11304–28

FIGURE 12. Transmit Counter XLT (Continued)



TL/L/11304-29

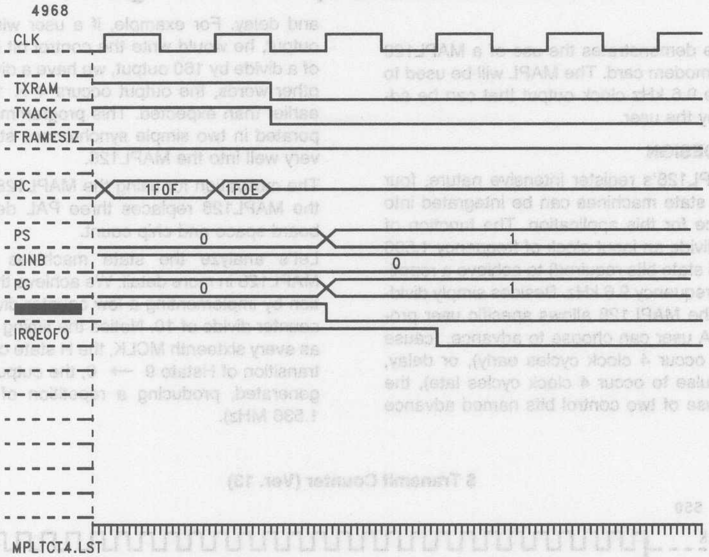
FIGURE 13. Timing Diagram



TL/L/11304-30

FIGURE 14. Timing Diagram

$ Transmit Counter (ver. 13)



TL/L/11304–31

FIGURE 15. Timing Diagram

SIGNAL DESCRIPTION

Inputs

FRAMESIZE — Determines the maximum size frame that may be transmitted.

0: 512 bytes

1: 8193 bytes

This is used by the MAPL design to determine the boundary crossing that signifies the end of a frame.

TXACK — Transmit acknowledge output from the NSC FDDI BMAC (DP83261) indicates that the transmitter is ready for the next data byte.

TXRAM ~ — Enables the MAPL design to address the Transmit RAM.

PC—ADDR(12:0) — These are the address inputs to the MAPL design. The address lines can be loaded and then incremented to address the Transmit RAM.

Outputs

MOREFRAMES — MOREFRAMES is asserted whenever there are more 512 byte frames in the Tx__RAM to be transmitted.

IRQEOF0 — Request end of frame. Indicates that the data ready to be transmitted is the last data byte when asserted. Normally, this is the last byte of the INFO field of the frame.

ADDROUT(12:0) — These are the latched PC__ADDR signals that address the Tx__RAM. The MAPL design takes the PC__ADDR, loads it, and then increments through the addresses until the Tx__RAM is empty.

7.0 A MAPL Divide Down Counter with Specific Timing Control Options

INTRODUCTION

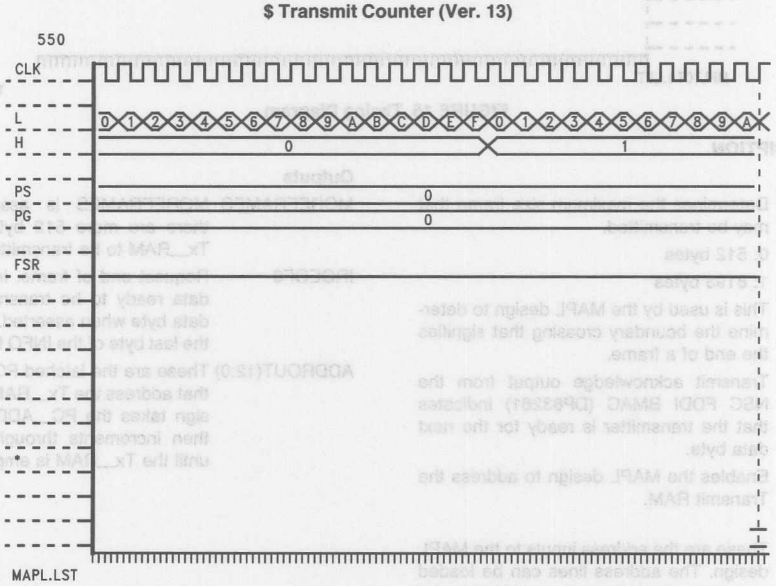
This application note demonstrates the use of a MAPL128 in a PC add-in FAX/modem card. The MAPL will be used to generate a repetitive 9.6 kHz clock output that can be advanced or delayed by the user.

DESCRIPTION OF DESIGN

Because of the MAPL128's register intensive nature, four simple synchronous state machines can be integrated into one MAPL128 device for this application. The function of the MAPL128 is to divide an input clock of frequency 1.536 MHz down by 160 (8 state bits required) to achieve a repetitive output clock of frequency 9.6 kHz. Besides simply dividing an input clock, the MAPL128 allows specific user programmable control. A user can choose to advance, (cause the output pulse to occur 4 clock cycles early), or delay, (cause the output pulse to occur 4 clock cycles late), the output through the use of two control bits named advance and delay. For example, if a user wishes to advance the output, he would write the control bit advance and instead of a divide by 160 output, we have a divide by 156 output. In other words, the output occurs 4 × 1.536 MHz or 2.6 μs earlier than expected. This programmable feature is incorporated in two simple synchronous state machines that fit very well into the MAPL128.

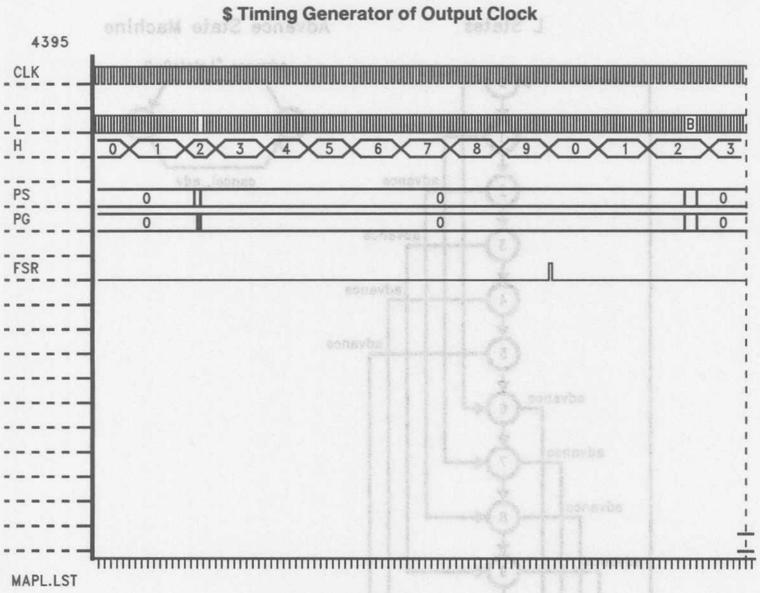
The motivation for using the MAPL128 is twofold. Because the MAPL128 replaces three PAL devices, it saves both board space and chip count.

Let's analyze the state machines that make up the MAPL128 in more detail: We achieve the divide by 160 function by implementing a low counter divide of 16 and a high counter divide of 10. Notice the timing diagram of *Figure 16* as every sixteenth MCLK, the H state changes and upon the transition of Hstate 9 ⟶ 0, the output pulse *(Figure 17)* is generated, producing a repetition of 9.6 kHz (1/160 × 1.536 MHz).



TL/L/11304-32

FIGURE 16. Timing Diagram



TL/L/11304-33

FIGURE 17. Timing Diagram

To advance the output, the user must write the control bit advance. Note the state diagram in *Figure 18* to see how the advance input causes a skipping of four states, thus achieving the necessary four clock cycle advance. Note also in *Figure 19* how a "cancel advance" signal is generated by the simple advance state machine. Cancel advance is necessary to clear the advance control bit written by the user. Without the cancel signal, the main counter state machine will keep skipping. Likewise, a user can realize a delaying of the output by writing the control bit delay. Note in *Figure 20* how "delay" causes a repeat of the same state, eventually staying in the same state four extra clocks to net a result output four clock cycles later. See how the delay feature repeats the state four times before it issues the CANCEL_DEL pulse which clears the delay control bit written by the user and allows the main counter state machine to continue.

The implementation of this design in a .OPL file can be seen in *Figure 21*. Since there are a few state machines in this design, the truth table function proved to be the simplest solution. The first section of the truth table implements the normal state transitions for the L states and the H states. This portion of the truth table also implements transitions from the normal state to advance or delay functions. The second portion of the truth table implements the advance function while the third portion implements the delay function.

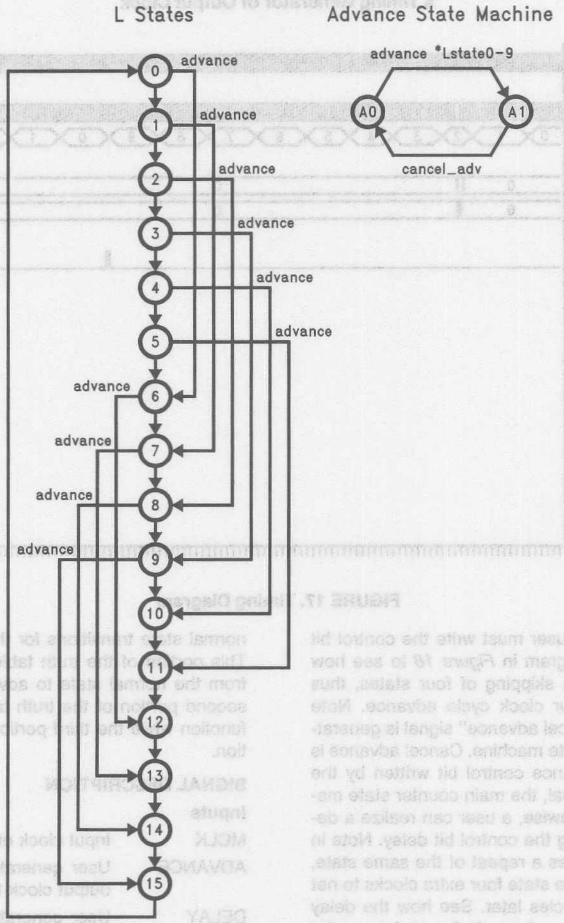
SIGNAL DESCRIPTION

Inputs

MCLK	Input clock of frequency 1.536 MHz.
ADVANCE	User generated control bit used to cause output clock to occur 4 MCLKs earlier.
DELAY	User generated control bit used to cause output clock to occur 4 MCLKs later.

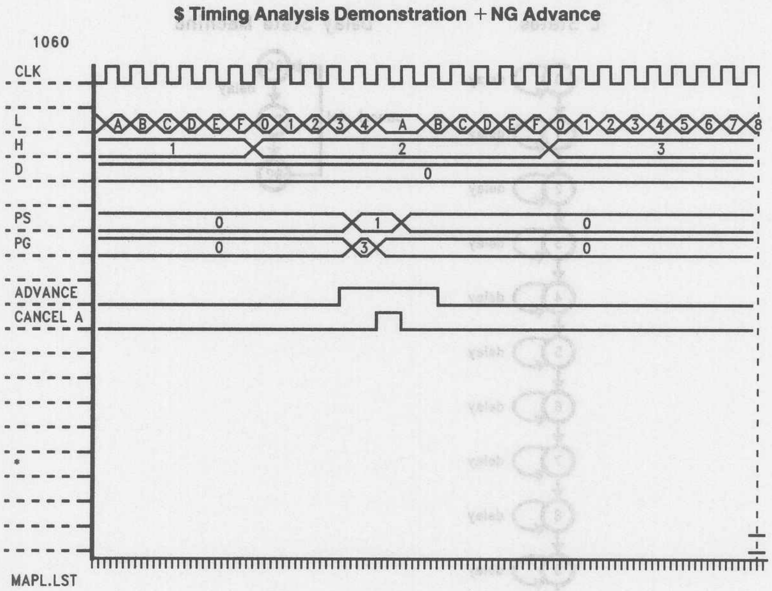
Outputs

FSR	Output clock of frequency 9.6 kHz.
CANCEL_ADV	Signal generated by MAPL to reset advance control bit.
CANCEL_DEL	Signal generated by MAPL to reset delay control bit.



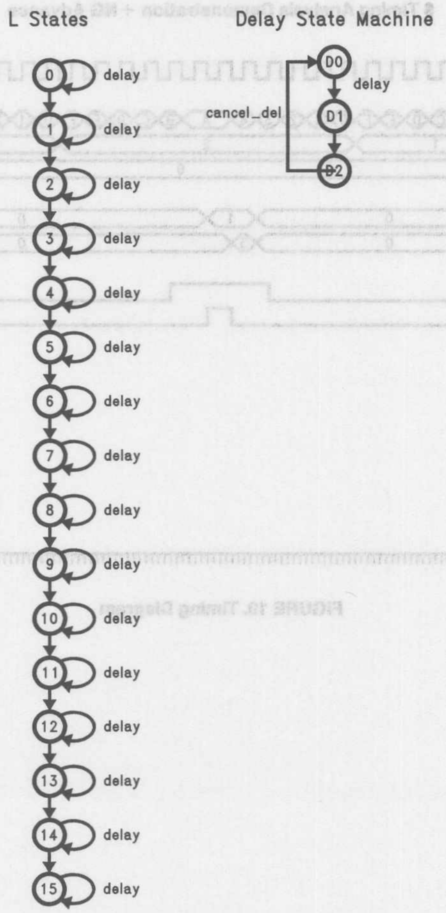
TL/L/11304-34

FIGURE 18. Advance State Machine



TL/L/11304–35

FIGURE 19. Timing Diagram



TL/L/11304-36

FIGURE 20. Delay State Machine

```
Begin Header
   Divide Down Counter with Specific Timing Control Options
End Header

Begin Definition
   Inputs
      rst, advance, delay ;
   Feedbacks (JK, HOLD)
      h3, h2, h1, h0, l3, l2, l1, l0, d2, d1, d0;
   Feedbacks (JK,RST)
      a0;
   Feedbacks (JK,RST,BURIED)
      g0, g1;
   Outputs
      cancel_adv, cancel_del, fsr ;
End Definition

Begin Equation
   global.re = rst;
End Equation

Begin Truth_Table
   ttin  advance, delay, h3, h2, h1, h0, l3, l2, l1, l0, a0, d2,
      d1, d0, g1, g0;
   ttout h3, h2, h1, h0, l3, l2, l1, l0, a0, d2, d1, d0,
      cancel_adv, cancel_del, fsr, g1, g0;

      -0-------- - ---00     --------! - --- --000
      -0-------1 - ---00     -------!- - --- --000
      -0------11 - ---00     ------!-- - --- --000
      -0-----111 - ---00     -----!--- - --- --000
      -0----1111 - ---00     ----!---- - --- --000
      -000011111 - ---00     0010---- - --- --000
      -00-101111 - ---00     0?11---- - --- --000
      -000111111 - ---00     0100---- - --- --000
      -001001111 - ---00     0101---- - --- --000
      -001011111 - ---00     0110---- - --- --000
      -001111111 - ---00     1000---- - --- --000
      -010001111 - ---00     1001---- - --- --000
      -010011111 - ---00     0000---- - --- --100
      10----0--- - ---00     -------- - --- --011
      10----100- - ---00     -------- - --- --011
      01-------- - ---00     -------- - 000 ---10
```

TL/L/11304-37

FIGURE 21. Divide Down Counter

```
      10----0000 0 ---11     ----0110 1 --- 1-000
      10----0001 0 ---11     ----0111 1 --- 1-000
      10----0010 0 ---11     ----1000 1 --- 1-000
      10----0011 0 ---11     ----1001 1 --- 1-000
      10----0100 0 ---11     ----1010 1 --- 1-000
      10----0101 0 ---11     ----1011 1 --- 1-000
      10----0110 0 ---11     ----1100 1 --- 1-000
      10----0111 0 ---11     ----1101 1 --- 1-000
      10----1000 0 ---11     ----1110 1 --- 1-000
      10----1001 0 ---11     ----1111 1 --- 1-000
      10----1010 0 ---11     ----1011 1 --- 1-000
      10----1011 0 ---11     ----1100 0 --- 0-000
      10----1100 0 ---11     ----1101 0 --- 0-000
      10----1101 0 ---11     ----1110 0 --- 0-000
      10----1110 0 ---11     ----1111 0 --- 0-000
      10----1111 0 ---11     ----0000 0 --- 0-000

      01-------- - 00010     --------- - 001 -0010
      01-------- - 00110     --------- - 011 -0010
      01-------- - 01110     --------- - 000 -1000

End Truth_Table
```

TL/L/11304-38

FIGURE 21. Divide Down Counter (Continued)

8.0 Bus Transaction and DMA Controller

This application demonstrates the use of the MAPL128 for bus state machine control. In this design, the MAPL128 provides bus access and DMA control for an I/O board on an asynchronous bus (Futurebus+). Although the design is fairly simple, the resulting state machine demonstrates flexibility and power of the OPAL command language. Less than half of the product terms of the MAPL128 are used in this example, so there is room for substantial increases in design sophistication.

The MAPL128 state machine controls accesses from the system bus to the local DRAM, from the local CPU to the system bus, and block data transfers between local DRAM and the system bus. The DMA controller is a discrete design, programmed by the CPU and enabled by the state machine. It consists of four sub-blocks:

1. The local address counter is programmed by the CPU before each DMA operation with the start address of the data block in DRAM. This can be either the source or destination of the transfer, depending on whether a bus write or read has been programmed.
2. The bus address counter is also programmed by the local CPU with the start address of the data block on Futurebus. Once again, this is independent of the direction of data transfer.
3. The transfer counter determines the total size of the DMA transfer. This is programmed by the local CPU at the start of each transfer. The local CPU can read the contents of the counter in the event of an error or unexpected DMA termination condition. Because this is an asynchronous bus, this counter must be clocked asynchronously as the DMA transfer progresses, and could not be implemented in the MAPL128.
4. The packet counter determines the number of words in each bus transaction, breaking up the DMA transfer into blocks that the bus can handle. The 8-bit counter allows packet sizes of up to 256 words (1 Kbytes on a 32-bit bus) in a single bus transaction. This is programmed by the local CPU as necessary; the counter is automatically reloaded with the last programmed value before each bus transfer.

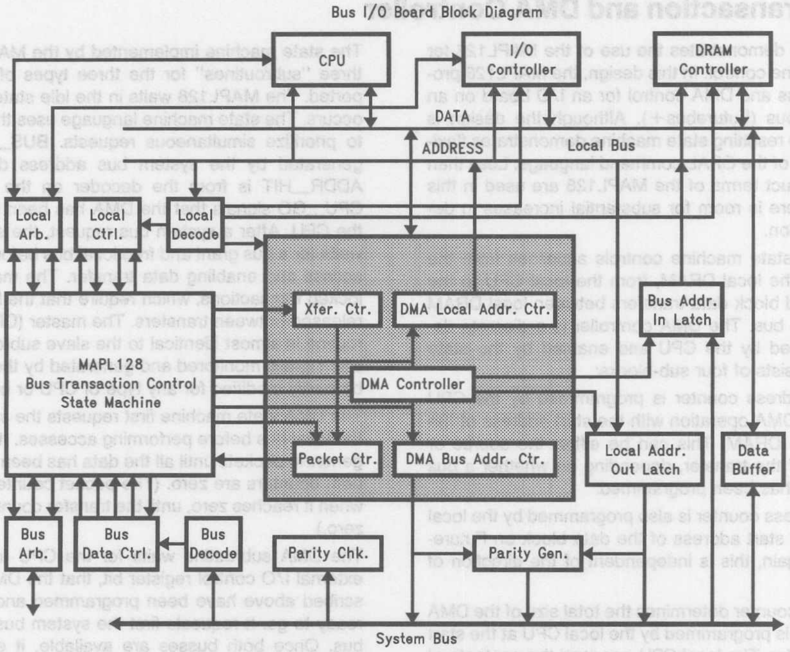
The MAPL128 state machine takes requests from the system bus and generates accesses into local DRAM, takes requests from the CPU and generates accesses out to the system bus, and, after the CPU has loaded the appropriate DMA controller registers, generates accesses to both the DRAM and the system bus. It enables the address latches for each type of access. It monitors the transfer and packet counters, so that multiple packets are generated until all the data is transferred. The actual data transfer is controlled by a separate asynchronous state machine.

The state machine implemented by the MAPL128 contains three "subroutines" for the three types of accesses supported. The MAPL128 waits in the idle state until a request occurs. The state machine language uses the if/else syntax to prioritize simultaneous requests. BUS_ADDR_HIT is generated by the system bus address decoder, CPU_ADDR_HIT is from the decoder on the local bus, and CPU_GO signals that the DMA has been programmed by the CPU. After a system bus request, the slave subroutine waits for a bus grant and free local bus before beginning the access and enabling data transfer. The machine supports locked transactions, which require that the local bus not be released between transfers. The master (CPU access) subroutine is almost identical to the slave subroutine. The specific signals monitored and generated by the MAPL128 can be easily modified for any type of CPU or bus interface.

The DMA state machine first requests the system bus, then the local bus before performing accesses. It will continue to generate packets until all the data has been transferred and both counters are zero. (The packet counter gets reloaded when it reaches zero, until the transfer counter also reaches zero.)

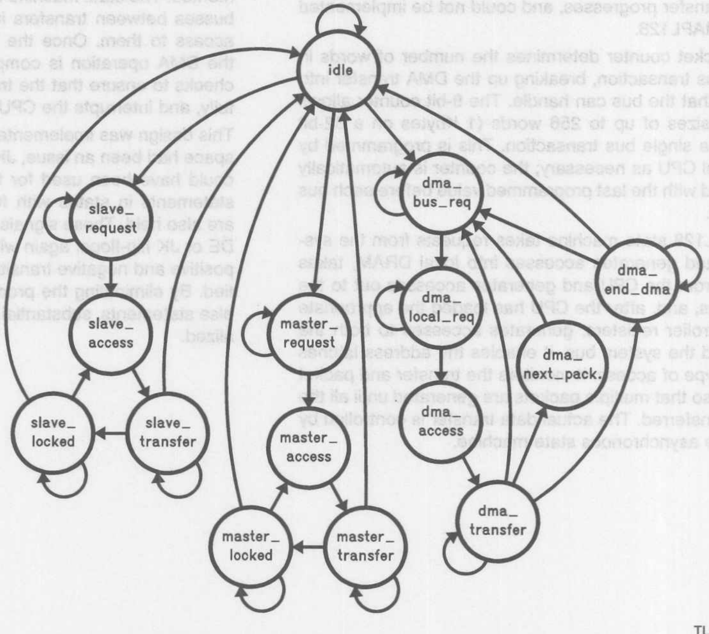
The DMA subroutine waits for the CPU to signal, via an external I/O control register bit, that the DMA registers described above have been programmed and the transfer is ready to go. It requests first the system bus, then the local bus. Once both busses are available, it enables the addresses onto both the local and system busses and starts the asynchronous bus transfer state machine. With each data transfer, the transfer and packet counters are decremented. The state machine will give up the local and system busses between transfers in order to allow other devices access to them. Once the transfer counter reaches zero, the DMA operation is complete. The DMA state machine checks to ensure that the transfer has completed successfully, and interrupts the CPU.

This design was implemented using D flip-flops only. If PLA space had been an issue, JK flip-flops with a default of hold could have been used for the state bits. The final "else" statements in states with feedback include signals which are also held. These signals would need to be converted to DE or JK flip-flops, again with a default of hold, and all the positive and negative transitions need to be explicitly specified. By eliminating the product terms associated with final else statements, substantial space savings can often be realized.



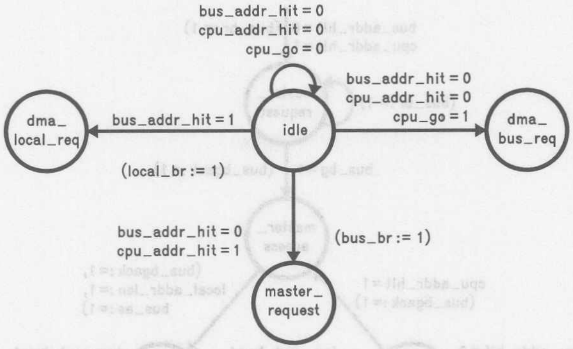
TL/L/11304-39

FIGURE 22. Bus I/O Board Block Diagram



TL/L/11304-40

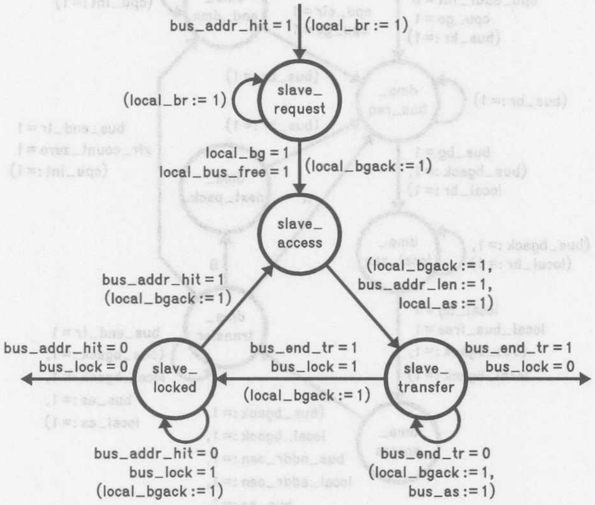
FIGURE 23. Bus Control State Machine



```
State idle:
 if bus_addr_hit then s_request
  with local_br := 1;
  endwith
 else
  if cpu_addr_hit then m_request
   with bus_br := 1;
   endwith
  else
   if cpu_go then d_bus_request
    with bus_br := 1;
    endwith
   else idle;
```

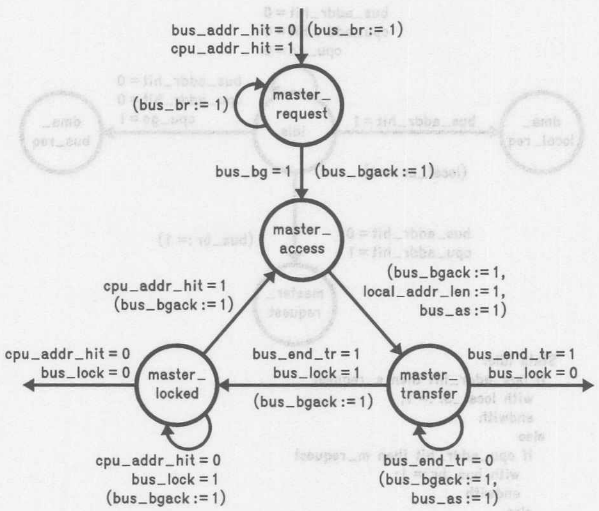
TL/L/11304-41

FIGURE 24. Bus Control State Machine (Idle State)



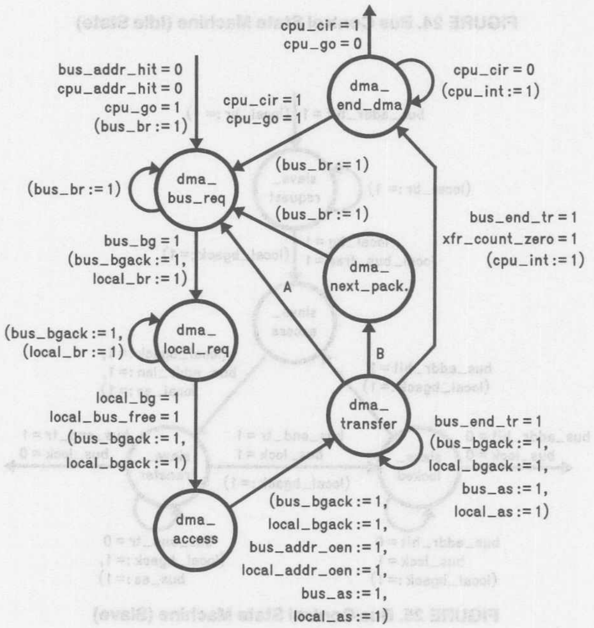
TL/L/11304-42

FIGURE 25. Bus Control State Machine (Slave)



TL/L/11304-43

FIGURE 26. Bus Control State Machine (Master)

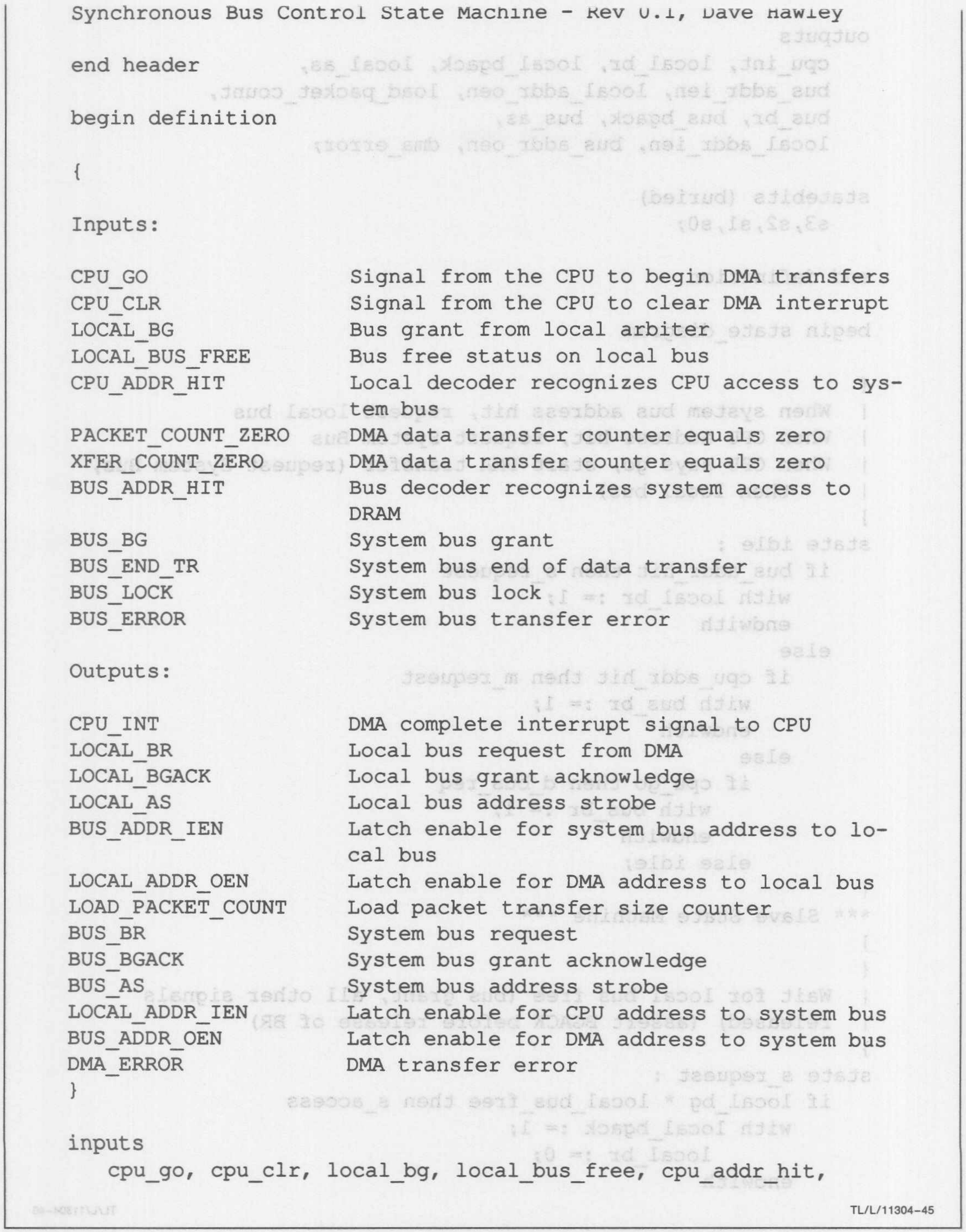


TL/L/11304-44

A: bus_end_tr = 1
packet_count_zero = 0
xfer_count_zero = 0
(bus_br: = 1)

B: bus_end_tr = 1
packet_count_zero = 0
xfer_count_zero = 0
(load_packet_count: = 1)

FIGURE 27. Bus Control State Machine (DMA)



```
Synchronous Bus Control State Machine - Rev 0.1, Dave Hawley

end header

begin definition

{

Inputs:

CPU_GO               Signal from the CPU to begin DMA transfers
CPU_CLR              Signal from the CPU to clear DMA interrupt
LOCAL_BG             Bus grant from local arbiter
LOCAL_BUS_FREE       Bus free status on local bus
CPU_ADDR_HIT         Local decoder recognizes CPU access to sys-
                     tem bus
PACKET_COUNT_ZERO    DMA data transfer counter equals zero
XFER_COUNT_ZERO      DMA data transfer counter equals zero
BUS_ADDR_HIT         Bus decoder recognizes system access to
                     DRAM
BUS_BG               System bus grant
BUS_END_TR           System bus end of data transfer
BUS_LOCK             System bus lock
BUS_ERROR            System bus transfer error

Outputs:

CPU_INT              DMA complete interrupt signal to CPU
LOCAL_BR             Local bus request from DMA
LOCAL_BGACK          Local bus grant acknowledge
LOCAL_AS             Local bus address strobe
BUS_ADDR_IEN         Latch enable for system bus address to lo-
                     cal bus
LOCAL_ADDR_OEN       Latch enable for DMA address to local bus
LOAD_PACKET_COUNT    Load packet transfer size counter
BUS_BR               System bus request
BUS_BGACK            System bus grant acknowledge
BUS_AS               System bus address strobe
LOCAL_ADDR_IEN       Latch enable for CPU address to system bus
BUS_ADDR_OEN         Latch enable for DMA address to system bus
DMA_ERROR            DMA transfer error
}

inputs
   cpu_go, cpu_clr, local_bg, local_bus_free, cpu_addr_hit,
```

TL/L/11304-45

```
   packet_count_zero, xfer_count_zero, bus_addr_hit,
   bus_bg, bus_end_tr, bus_lock, bus_error;
outputs
   cpu_int, local_br, local_bgack, local_as,
   bus_addr_ien, local_addr_oen, load_packet_count,
   bus_br, bus_bgack, bus_as,
   local_addr_ien, bus_addr_oen, dma_error;

statebits (buried)
   s3,s2,s1,s0;

end definition

begin state_diagram

{
|  When system bus address hit, request local bus
|  When CPU address hit, request System Bus
|  When CPU says go, start DMA transfer (request System Bus,
|     then local bus)
}
state idle :
   if bus_addr_hit then s_request
      with local_br := 1;
      endwith
   else
      if cpu_addr_hit then m_request
         with bus_br := 1;
         endwith
      else
         if cpu_go then d_bus_req
            with bus_br := 1;
            endwith
         else idle;
{
*** Slave State Machine ***
}
{
|  Wait for local bus free (bus grant, all other signals
|  released) (assert BGACK before release of BR)
}
state s_request :
   if local_bg * local_bus_free then s_access
      with local_bgack := 1;
            local_br := 0;
      endwith
```

TL/L/11304–46

```
   else s_request
      with local_br := 1;
      endwith;
{
|  Begin bus transaction
|  (assert address enable before AS)
}
state s_access :
   goto s_transfer
      with local_bgack := 1;
         bus_addr_ien := 1;
         local_as := 1;
      endwith;
{
|  Can release the address enable immediately (latched by
|     DRAM controller)
|  When transaction complete, release AS
|  If no lock, release local bus, otherwise enter locked state
}
state s_transfer :
   case bus_end_tr * /bus_lock : idle;
         bus_end_tr * bus_lock : s_locked
            with local_bgack := 1;
            endwith;
         /bus_end_tr : s_transfer
            with local_bgack := 1;
               bus_addr_ien := 0;
               local_as := 1;
            endwith;
   endcase;
{
|  In locked state, wait for new address hit or lock release
}
state s_locked :
   case bus_addr_hit : s_access
         with local_bgack := 1;
         endwith;
      /bus_addr_hit * /bus_lock : idle;
      /bus_addr_hit * bus_lock : s_locked
         with local_bgack := 1;
         endwith;
   endcase;
{
*** Master State Machine ***
}
{
| Wait for system bus grant
```

TL/L/11304-47

```
| (assert BGACK before release of BR)
}
state m_request :
  if bus_bg then m_access
    with bus_bgack := 1;
        bus_br := 0;
    endwith
  else m_request
    with bus_br := 1;
    endwith;
{
| Begin bus transaction
| (assert address enable before AS)
}
state m_access :
  goto m_transfer
    with bus_bgack := 1;
      local_addr_ien := 1;
      bus_as := 1;
    endwith;
{
| Can release the address enable immediately
| When transaction complete, release AS
| If no lock, release System Bus, otherwise enter locked state
}
state m_transfer :
  case bus_end_tr * /bus_lock : idle;
      bus_end_tr * bus_lock : m_locked
        with bus_bgack := 1;
        endwith;
      /bus_end_tr : m_transfer
        with bus_bgack := 1;
          local_addr_ien := 0;
          bus_as := 1;
        endwith;
  endcase;
{
| In locked state, wait for new address hit or lock release
}
state m_locked :
  case cpu_addr_hit : s_access
      with bus_bgack := 1;
      endwith;
    /cpu_addr_hit * /bus_lock : idle;
    /cpu_addr_hit * bus_lock : s_locked
      with bus_bgack := 1;
      endwith;
```

TL/L/11304–48

```
    endcase;
{
*** DMA State Machine ***
}
{
| Wait for System Bus grant, then request local bus
| (assert BGACK before release of BR)
}
state d_bus_req :
  if bus_bg then d_local_req
    with bus_bgack := 1;
      bus_br := 0;
      local_br := 1;
    endwith
  else d_bus_req
    with bus_br := 1;
    endwith;
{
| Wait for local bus free (bus grant, all other signals released)
| (assert BGACK before release of BR)
}
state d_local_req :
  if local_bg * local_bus_free then d_access
    with bus_bgack := 1;
        local_bgack := 1;
        local_br := 0;
    endwith
  else d_local_req
    with bus_bgack := 1;
      local_br := 1;
    endwith;
{
| Begin DMA transaction
| (assert address enables before AS)
}
state d_access :
  goto d_transfer
    with bus_bgack := 1;
        local_bgack := 1;
        bus_addr_oen := 1;
        local_addr_oen := 1;
        bus_as := 1;
        local_as := 1;
    endwith;
{
| Can release the address enables immediately
| When transaction complete, release AS
```

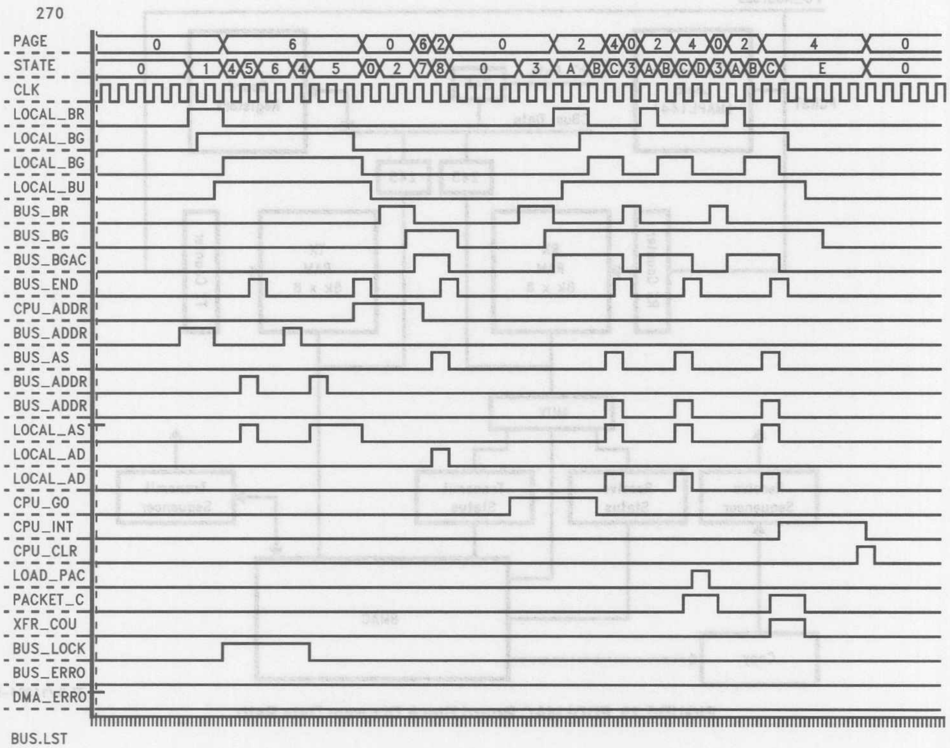
TL/L/11304-49

```
| Interrupt CPU, reload packet counter, or re-request bus,
|   depending on packet and transfer counter states
}
state d_transfer :
  case bus_end_tr * /bus_error * /packet_count_zero *
          /xfer_count_zero :
      d_bus_req
        with bus_br := 1;
        endwith;
      bus_end_tr * /bus_error * packet_count_zero *
          /xfer_count_zero :
      d_next_packet
        with load_packet_count := 1;
        endwith;
      bus_end_tr * /bus_error * packet_count_zero *
          xfer_count_zero :
      d_end_dma
        with cpu_int := 1;
        endwith;
      bus_end_tr * /bus_error * /packet_count_zero *
          xfer_count_zero :
      d_end_dma
        with cpu_int := 1;
          dma_error := 1;
        endwith;
      /bus_end_tr : d _transfer
        with bus_bgack := 1;
          local_bgack := 1;
          bus_addr_oen := 0;
          local_addr_oen := 0;
          bus_as := 1;
          local_as := 1;
        endwith;
  endcase;
{
| Transfer count <> 0, so reload packet counter and continue
}
state d_next_packet :
  goto d_bus_req
    with load_packet_count := 0;
        bus_br := 1;
    endwith;
{
| Error or transfer count = 0 so interrupt CPU and
| wait for clear interrupt
}
{
```

TL/L/11304-50

```
state d_end_dma :
  case cpu_clr * /cpu_go : idle;
       cpu_clr * cpu_go : d_bus_req
         with bus_br :- 1;
         endwith;
       /cpu_clr * cpu_go : d_end_dma
         with cpu_int := 1;
              dma_error := 1;
         endwith;
       /cpu_clr * /cpu_go : d_end_dma
         with cpu_int := 1;
         endwith;
  endcase;

end state_diagram
```

TL/L/11304-51



TL/L/11304-52

FIGURE 28. Timing Diagram

9.0 A MAPL PC Interface Module for the FDDI MAC Layer Evaluation Board

INTRODUCTION

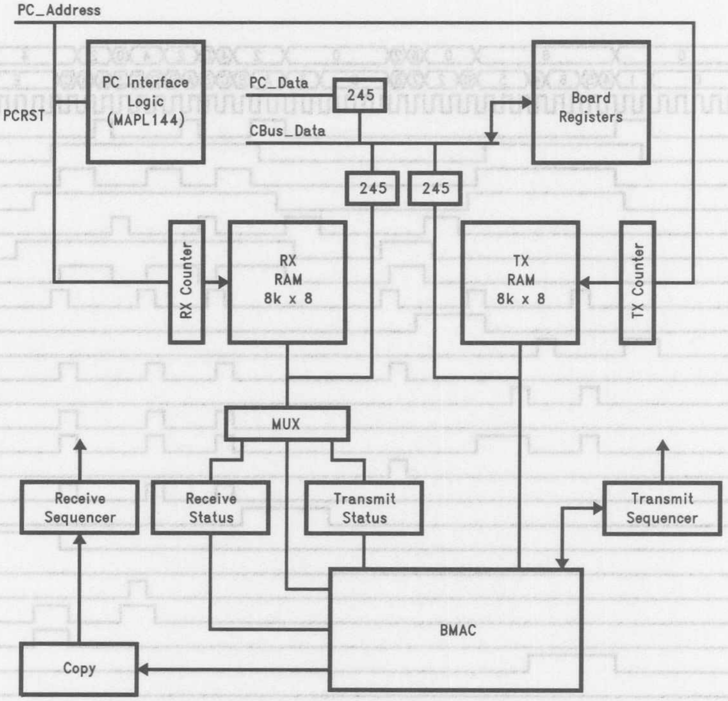
This application uses the MAPL144, Multiple Array Programmable Logic device, as a PC interface Module. The MAPL144 provides access to the transmit and receive RAMs in addition to board registers, and the BMAC peripheral chip. Since the MAPL144 is a sequential device, the decode logic is based on a state-machine which controls accesses from the system bus. OPAL software is used to compile the high-level .OPL file into a JEDEC map which can be used with Viewlogic® or OPALSIM to test and verify the design. It is assumed that the reader is already familiar with the FDDI MAC Layer Evaluation Board.

DESCRIPTION OF DESIGN

Figure 29 gives a simple block diagram of the FDDI MAC Board Data Path, while *Figure 30* gives a block diagram of the Control Logic. The function of the PC interface Module is to interface the FDDI MAC Layer Evaluation Board to the PC host. This block features a 20-bit address bus for flexible memory map placement. In addition to the address bus, five PC bus control signals as well as a couple of board interface signals are included.

The PC interface Module provides the chip selects and intermediate signals needed for memory select. The board registers MODE, STATUS, and FUNCTION are all controlled by the PC interface Module. The peripheral BMAC chip is also controlled by the PC interface with the signal BMACSEL. The intermediate signals generated by the PC interface for memory control are MEMSEL and ACK3. *Figure 31* shows the address mapping for the specific functions on the board.

The PC interface currently used on the FDDI MAC Layer Evaluation Board utilizes four GAL20V8's. The entry files for these four GAL® devices were obtained using Boolean equations. Three of the GAL devices can be integrated into the MAPL144. The fourth GAL device provides combinatorial access of the memories on the board.

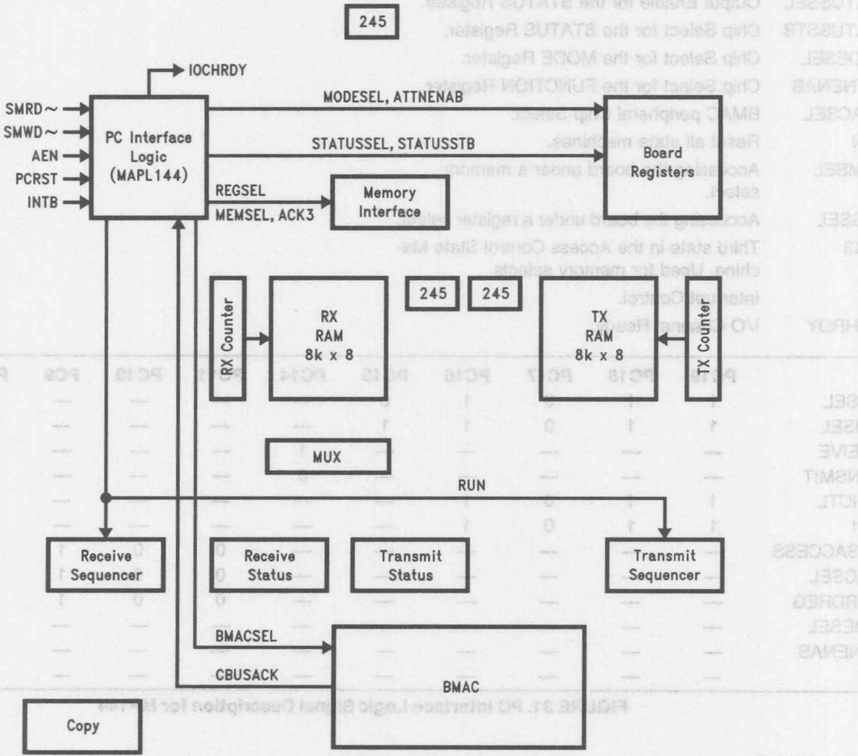


TL/L/11304-53

FIGURE 29. FDDI MAC Board Block Diagram Data Path

When the Boolean equations for the three GAL devices are combined into one .OPL file, the logic will not compile into the device because of logic partitioning problems. In other words, it is unable to allocate logic to different pages when it is specified as being on one page. The PC interface was redesigned based on a state machine so as to split up the decode logic. Most of the decode logic in the Boolean equations was based on access feedback terms. These access feedback terms could be incorporated into a state machine. This access control state machine is illustrated in *Figure 32*. Now, each of the eight states can be placed on one page of the MAPL144, thus partitioning the decode logic.

Incorporating the state machine into the .OPL file is now trivial. *Figure 33* shows the final .OPL file for this design. Notice how the page bits are defined as state bits in the definition block. This allows the designer to free-up three pins for designs that are I/O limited. *Figure 34* shows the output from the fitter. In this particular design, manual paging was done by using the page bits as state bits. This way, the designer has the freedom to partition the design the way he wants it to be partitioned.



TL/L/11304–54

FIGURE 30. FDDI MAC Board Block Diagram Control Logic

Signal Description

Inputs

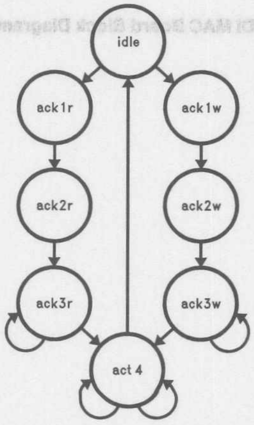
SMWR ~	System write.
SMRD ~	System read.
AEN ~	Address Enable.
PCRST	PC Reset.
CBUSACK	CBUS Acknowledge from BMAC peripheral.
INTB	Inverse Interrupt Signal.
CLK	System Clock.

Outputs

STATUSSEL	Output Enable for the STATUS Register.
STATUSSTB	Chip Select for the STATUS Register.
MODESEL	Chip Select for the MODE Register.
ATTNENAB	Chip Select for the FUNCTION Register.
BMACSEL	BMAC peripheral Chip Select.
RUN	Reset all state machines.
MEMSEL	Accessing the board under a memory select.
REGSEL	Accessing the board under a register select.
ACK3	Third state in the Access Control State Machine. Used for memory selects.
INT	Interrupt Control.
IOCHRDY	I/O Channel Ready.

	PC19	PC18	PC17	PC16	PC15	PC14	PC11	PC10	PC9	PC8	PC7	PC6
REGSEL	1	1	0	1	0	—	—	—	—	—	—	—
MEMSEL	1	1	0	1	1	—	—	—	—	—	—	—
RECEIVE	—	—	—	—	—	1	—	—	—	—	—	—
TRANSMIT	—	—	—	—	—	0	—	—	—	—	—	—
IOCHCTL	1	1	0	1	—	—	—	—	—	—	—	—
ACK1	1	1	0	1	—	—	—	—	—	—	—	—
CBUSACCESS	—	—	—	—	—	—	0	0	1	0	—	—
BMACSEL	—	—	—	—	—	—	0	0	1	0	—	—
BOARDREG	—	—	—	—	—	—	0	0	1	1	—	—
MODESEL	—	—	—	—	—	—	—	—	—	—	0	0
ATTNENAB	—	—	—	—	—	—	—	—	—	—	0	1
RUN	—	—	—	—	—	—	—	—	—	—	1	1

FIGURE 31. PC Interface Logic Signal Description for MP144



TL/L/11304-55

FIGURE 32. PC Interface Access Control State Machine

```
begin header
   pc interface (state machine)
end header

begin definition
   device MAPL144;

   inputs
      aen,smwr,smrd,
      pc19,pc18,pc17,pc16,pc15,pc11,pc10,pc9,pc8,
      pc7,pc6,pc5,pcrst,cbusack,intb;
   outputs (JK,RST)
      statussel,modesel,attnenab,run,bmacsel,ack3,memsel,
      regsel,statusstb,int,iochrdy;
   statebits (DE,RST,BURIED)
      pb2=47,pb1=46,pb0=45;
   STATE_NAME
      IDLE=^b000,ACK1W=^b001,ACK1R=^b100,ACK2W=^b010,
      ACK2R=^b101,ACK3W=^b011,ACK3R=^b110,ACK4RW=^b111;

end definition

begin equation

   int := /intb;

end equation

begin state_diagram
   state IDLE :
      CASE
      /smrd*/aen*pc19*pc18*/pc17*pc16 : ACK1R
         with /iochrdy := 1; endwith;
      /smwr*/aen*pc19*pc18*/pc17*pc16 : ACK1W
         with /iochrdy := 1; endwith;
      ENDCASE;

   state ACK1W :
      goto ACK2W
      with
         modesel := /pc15*/pc11*/pc10*pc9*pc8*/pc7*/pc6;
```

TL/L/11304–56

FIGURE 33. PC Interface .OPL File

```
      attnenab := /pc15*/pc11*/pc10*pc9*pc8*/pc7*pc6;
      statussel := (pc7*/pc6*/pc5*/pc15*/pc11*/
        pc10*pc9*pc8);
      statusstb := pc7*/pc6*/pc5;
      run := pc7*pc6*/pc15*/pc11*/pc10*pc9*pc8+pcrst;
      bmacsel := /pc15*/pc11*/pc10*pc9*/pc8;
      regsel := /pc15;
      memsel := pc15;
      /iochrdy := 1;
    endwith;

  state ACK1R :
    goto ACK2R
    with
      modesel := /pc15*/pc11*/pc10*pc9*pc8*/pc7*/pc6;
      attnenab := /pc15*/pc11*/pc10*pc9*pc8*/pc7*pc6;
      statussel := (pc7*/pc6*/pc5*/pc15*/pc11*/
        pc10*pc9*pc8);
      statusstb := pc7*/pc6*/pc5;
      run := pc7*pc6*/pc15*/pc11*/pc10*pc9*pc8+pcrst;
      bmacsel := /pc15*/pc11*/pc10*pc9*/pc8;
      regsel := /pc15;
      memsel := pc15;
      /iochrdy := 1;
    endwith;

  state ACK2W :
    goto ACK3W
    with
      modesel := ?;
      attnenab := ?;
      statussel := ?;
      statusstb := ?;
      run := ?;
      bmacsel := ?;
      regsel := ?;
      memsel := ?;
      ack3 := 1;
      /iochrdy := 1;
    endwith;
```

TL/L/11304–57

FIGURE 33. PC Interface .OPL File (Continued)

```
state ACK2R :
   goto ACK3R
   with
      modesel := ?;
      attnenab := ?;
      statussel := ?;
      statusstb := ?;
      run := ?;
      bmacsel := ?;
      regsel := ?;
      memsel := ?;
      ack3 := 1;
      /iochrdy := 1;
   endwith;

state ACK3W :
   if (pc15+pc11+pc10+/pc9+pc8)+
                 (/pc15*/pc11*/pc10*pc9*/pc8*/cbusack)
   then ACK4RW
   else ACK3W
   with
      modesel := ?;
      attnenab := ?;
      statussel := ?;
      statusstb := ?;
      run := ?;
      bmacsel := ?;
      regsel := ?;
      memsel := ?;
      ack3 := 1;
      /iochrdy := 1;
   endwith;
```

TL/L/11304-58

FIGURE 33. PC Interface .OPL File (Continued)

```
                   (/pc15*/pc11*/pc10*pc9*/pc8*/cbusack)
      then ACK4RW
      with
         statussel := ?;
         run := ?;
         bmacsel := ?;
         regsel := ?;
         memsel := ?;

      endwith
      else ACK3R
      with
         modesel := ?;
         attnenab := ?;
         statussel := ?;
         statusstb := ?;
         run := ?;
         bmacsel := ?;
         regsel := ?;
         memsel := ?;
         ack3 := 1;
         /iochrdy := 1;
      endwith;

   state ACK4RW :
      if /smrd then ACK4RW
      with
         statussel := ?;
         run := ?;
         bmacsel := ?;
         regsel := ?;
         memsel := ?;
      endwith
      else if /smwr then ACK4RW
      else IDLE
      with statussel := 1; endwith;

end state_diagram
```

TL/L/11304-59

FIGURE 33. PC Interface .OPL File (Continued)

10.0 PLUS405 to MAPL™128 Conversion

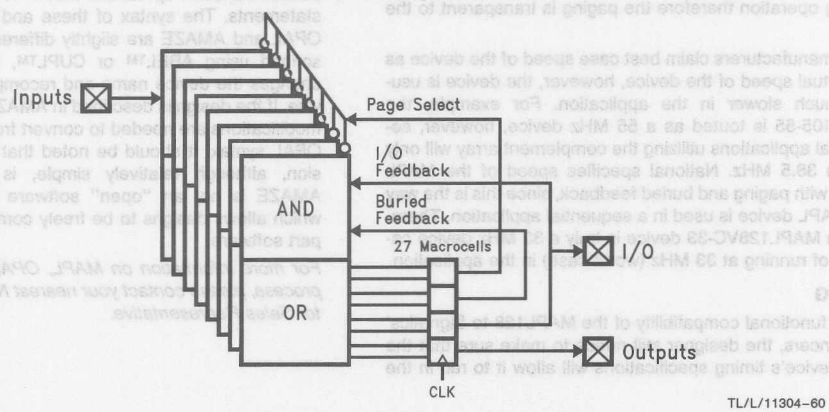
INTRODUCTION

Many state machine designers are familiar with Philips/Signetics sequencers for state machine design and will appreciate the density, low power, and speed of National Semiconductor's MAPL128. The MAPL128 in 28-pin PLCC has the speed and density to tackle many existing and future designs including many current Signetics' sequencer designs. In fact, in many cases the MAPL128 is pin-for-pin compatible with these previous designs while offering the designer higher speed, lower power, and more logic capacity. The following devices are available in 28-pin PLCC:

GENERAL COMPARISON

The MAPL128 and the 28-pin Signetics sequencers have very similar pinouts. Thus, the MAPL128 device can usually be dropped into the socket for evaluation. In some cases, a couple of pins must be swapped. It should be noted that the MAPL128 has significantly more logic capacity than the other devices, allowing the designer to not only replace the old device, but improve (and add to) the design. In fact, upgrading to a MAPL128 device will not only increase functionality, but dramatically reduce power consumption. For example, the PLUS405 device consumes 225 mA, more than twice the power of MAPL128 which consumes 110 mA, while the MAPL128 has 40% more logic capacity than the PL05405. National reduces power consumption by using EECMOS versus bipolar, and more importantly, by implementing a revolutionary paged architecture, giving MAPL devices lower power consumption and higher logic capacity without the usual speed reduction.

MAPL128 Block Diagram



TL/L/11304-60

Mfgr	Device	In	Out	I/O	Buried	Prod Terms	I_{CC}	t_{SU} (Note 1)	t_{CLK} (Note 1)	f_{MB} (Note 1)	Repl (Note 2)
NSC	MAPL128	9	4	12	8	128	110	17	9	40	—
Sig	PLUS405 (Note 3)	16	8	0	8	64	225	18	8	38.5	Y
Sig	PLUS105 (Note 3)	16	8	0	6	48	200	20	8	35.7	Y
Sig	PLC42VA12 (Note 3)	8	2	10	10	64	120	50	17	13.3	N
Sig	PLC415 (Note 3)	17	8	0	8	68	75	60	22	13.3	Y
Sig	PLS179 (Note 3)	8	4	8	0	48	210	55	20	13.3	N
Sig	PLS168/A (Note 3)	12	4	4	6	48	180	70	20	12.5	Y
Sig	PLS167/A (Note 3)	14	4	2	6	48	180	70	20	12.5	Y
Sig	PLS105/A (Note 3)	16	8	0	6	48	180	70	20	12.5	Y

Note 1: Worst case speed: with buried feedback through complement array/page terms.

Note 2: Pinout same as MAPL128. PLUS405 and PLC415 are replaceable if pin 4 is not used as clock. PLUS 168/A are replaceable if pins 15 & 16 are used as outputs. PLUS105/A are replaceable if pin 16 is used as an output. PLUS105 and PLS105/A are pin for pin replaceable by MAPL128.

Note 3: Specifications are taken from fastest version of the device from the 1991 databook. Specifications are subject to change.

PAGING VS COMPLEMENT ARRAY

National's paged architecture might appear similar to Signetics complement array, but it is far different. Signetics feeds back one or two OR array output signals through a NOR gate directly back into the AND array. They call these feedback signal the complement array, which are used to more effectively use product terms in state machine designs. National feeds three signals and their complements into the AND array that not only can be used to perform a "complement array" function, but also to select one of eight pages. These page select signals are hard-wired so that one and only one page is active at a time. This reduces power consumption and increases device performance. At the time of compiling the design, "fitter" software automatically partitions the logic across the pages of the MAPL device and assigns page bits to each state. On each clock cycle the correct page is selected based on where the next logic state can be found. There is no time penalty for this paging operation therefore the paging is transparent to the user.

Most manufacturers claim best case speed of the device as the actual speed of the device, however, the device is usually much slower in the application. For example, the PLUS405-55 is touted as a 55 MHz device, however, sequential applications utilizing the complement array will only realize 38.5 MHz. National specifies speed of the MAPL family with paging and buried feedback, since this is the way the MAPL device is used in a sequential application. Therefore, a MAPL128VC-33 device is truly a 33 MHz device capable of running at 33 MHz (worst case) in the application.

TIMING

Given functional compatibility of the MAPL128 to Signetics' sequencers, the designer still needs to make sure that the new device's timing specifications will allow it to run in the system. In state machine design, t_{SU} (setup time) and t_{CLK} (time of clock to registered output), are very important. Careful attention should be paid to these parameters to be sure that a device will function properly in the application.

Device	t_{SU}	t_{CLK}	f_{MB}
MAPL128	17	9	40
	20	10	33
PLUS4O5	18	8	38.5
	22	12	29.4
	25	15	25

PLD SOFTWARE COMPATIBILY

Both OPAL™ software from National and AMAZE software from Signetics allow state machine designs to be efficiently written in a state machine language. This language describes transition opreations with IF-THEN-ELSE and CASE statements. The syntax of these and other statements in OPAL and AMAZE are slightly different. If a design is described using ABEL™ or CUPL™, the designer simply changes the device name and recompiles for the new device. If the design is described in AMAZE, then some simple modifications are needed to convert from AMAZE syntax to OPAL syntax. It should be noted that this manual conversion, although relatively simple, is necessary becase AMAZE is not an "open" software package like OPAL which allows designs to be freely communicated with third part software.

For more information on MAPL, OPAL, or the conversion process, please contact your nearest National Semiconductor Sales Representative.

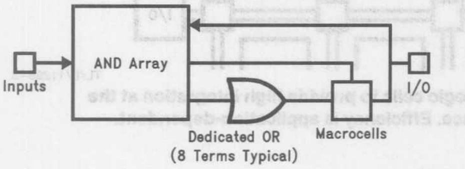
INTRODUCTION

The range of programmable logic devices can be divided into three broad architectural areas: PALs, PLAs, and FPGAs. Although designers probably wish there was a single architecture suitable for all applications, in reality each has its area of specialty. National Semiconductor's new MAPL™ family, for example, is a high-speed, PLA architecture ideal for state machine applications. Neither PALs nor FPGAs can provide the same performance and ease of use over a broad range of design complexity. This paper analyzes the application strengths and weaknesses of the latest PLD architectures, then focuses on the specific requirements of state machine design.

THE PAL® ARCHITECTURE

The basic PLD architecture is the PAL, with a programmable AND array and a fixed number of independent product terms per output *(Figure 1)*. Although many refinements have been made to the output macrocells and distribution of product terms, the basic architecture remains the same. Its high speed, high fan-in, and moderate number of product terms per output make it ideal for the largest segments of the PLD market: random logic replacement and address decoding. Indeed, the GAL® 22V10 architecture is the most popular PLD today, and similar devices are available from a wide range of vendors.

The biggest advantage of the PAL architecture is its high speed; the biggest limitation is its relatively low density. Unfortunately, it is impossible to simply scale-up the PAL architecture without sacrificing performance and power. By examining characteristics of the target applications, however, manufacturers have been able to find an acceptable compromise. Since most random logic uses a small number of product terms, a device with a large number of I/O macrocells, each with only a few product terms, can provide a much higher effective density. There are several partitioned-PAL architectures on the market now which aim for the random logic replacement applications.



TL/L/11296–1

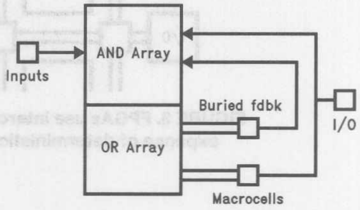
FIGURE 1. The PAL architecture uses a programmable AND array and dedicated OR array for high speed implementation of random logic and decode functions.

Even random logic requires the occasional high-product term equation. Those architectures which solve this problem with the least disturbance to the output performance are generally the easiest to work with. If adding a product term to an equation changes the setup time of a signal by 10 ns, the performance of the entire system may have to be reduced. Folded NAND or expander arrays have this characteristic, and make a preliminary timing analysis of any design very difficult. Likewise, the interconnect between PAL arrays within a single chip can hide limitations until a final compilation of the design into the chip is attempted and design modification may cause unwanted pinout changes.

High-density PAL architectures have all made certain compromises in order to maintain performance. As long as the designer is aware of details of the architecture and how they relate to the characteristics of the application, they can be quite useful in reducing system part count. A software package which supports more than one manufacturer's device can be a useful evaluation tool. For raw speed and simplicity, however, standard PAL and GAL devices will always be at the leading edge.

THE PLA ARCHITECTURE

The PLA architecture differs from the PAL in that both the AND array and OR array are programmable *(Figure 2)*. This means that all product terms are available to all outputs: several outputs can share the same product term, and each output can have a large number of product terms. As a result, the utilization of product terms in a PLA is generally much higher than in a PAL. The high fan-in and large number of product terms make PLAs ideal for state machine design, the third largest PLD application. Unfortunately, PLAs are generally much slower than PALs as well, since each signal must traverse two arrays. This means that designers have tended to avoid PLAs in speed-sensitive applications.



TL/L/11296–2

FIGURE 2. The traditional PLA architecture, ideal for state machines, uses two programmable arrays for high density at the expense of speed.

Other, non-PLA solutions to state machine design have been attempted. These include various microcoded sequencers and pipelined arrays. However, they have sacrificed the basic simplicity of the sum-of-products architecture. Along with adding their own limitations, they lose some of the other benefits of a PLA. One of these is product term sharing, which allows PLA devices additional freedom in macrocell design. In a PAL, each macrocell input requires its own product terms, which impacts density, and therefore performance. In a PLA, an increase in the number of macrocell inputs does not require a corresponding increase in the number of product terms. Instead of only D flip-flops or latches, a PLA macrocell can be built with SR, JK, and DE flip-flops for additional design efficiency.

The biggest advantage of the PLA architecture is its high density; the biggest limitation is its low speed. Once again, it is possible to examine the characteristics of the target application to find a way around the performance limitation. Although most state machines require a large number of product terms, only a few of them are needed by the current state. In a paged-PLA architecture, a subset of the total array product terms are powered up at any given time. The power saved goes into making the device fast. National Semiconductor is the first manufacturer to provide an architectural solution to PLA state machine design.

THE FPGA ARCHITECTURE

FPGAs are the latest development in the area of programmable logic. Although the technology used to implement them may vary, and manufacturers of high-density PALs and PLAs often mislabel their devices FPGAs, they are really quite easy to define. Instead of programmable AND or OR arrays, they have an array of programmable logic cells, interconnected by a programmable wiring matrix *(Figure 3)*. These logic cells have limited fan-in and fan-out, and the internal routing results in variable delays through the device. Their performance and efficiency tends to be highly application-dependent.

FPGAs are ideal for high-density logic replacement, where the performance of PLDs is not required. Because of the low fan-in, they are best where the complexity of each logic block in the design is small. Unlike PALs and PLAs, with close to full global interconnect, their limited routing resources make FPGAs better for designs with more local than globally routed signals. These characteristics lend themselves to the design of subsystems for which no custom silicon exists, and which would otherwise have to be built from discrete parts.

Unfortunately, it is difficult to know how well a design will match a particular FPGA architecture without a lot of design experience or a preliminary implementation. Due to the cost of FPGA design software, a large investment is often required before an estimate of efficiency, utilization, and system timing can be obtained. With respect to the main application areas of traditional PLDs, however, a few gross generalizations can be drawn. FPGAs work well for random logic replacement, if they can meet the propagation delay requirements of the design. They tend to be too slow and awkward for address decoding; the regular structure of a PAL is a much better match. And their performance degrades rapidly with the complexity of state machines, due to the high level of fan-in and interconnectivity that is typically required. A high-speed PLA architecture is much easier to work with.

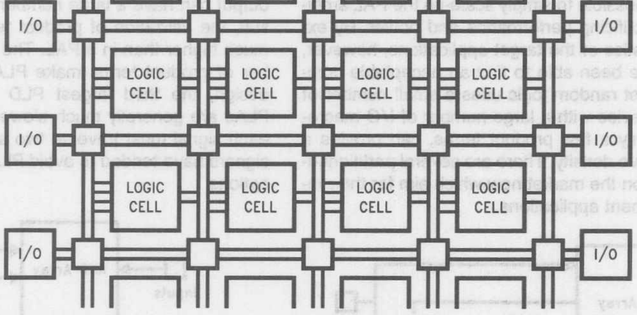


FIGURE 3. FPGAs use interconnected logic cells to provide high integration at the expense of deterministic performance. Efficiency is application-dependent.

THE MAPL PLA ARCHITECTURE

The MAPL is targeted at high-speed state machine applications. This is a broad area which includes local and system bus interfaces, arbitration, timing control, micro sequencers, and almost any other type of synchronous (registered) logic design. All of these benefit from the full interconnect and consistent timing provided by a PLA architecture. The paged PLA architecture of the MAPL gives the user access to the largest, fastest, and lowest power EECMOS PLA on the market *(Figure 4)*.

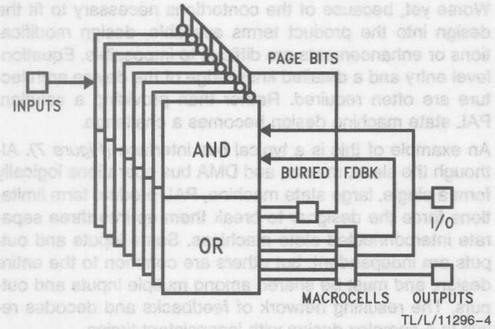


FIGURE 4. The MAPL paged PLA architecture allows implementation of complex state machines without sacrificing performance or power.

The MAPL array provides the user with 128 product terms. This is the largest PLA on the market, and allows unprecedented design complexity in a traditional architecture. The flexible output macrocells are designed to increase product term efficiency still further. It supports a 45 MHz true system clock rate, which is higher than most bipolar PLAs. The 8 ns clock to output specification is the best of any high-density PLD. And the maximum device current, for any design, is specified at 110 mA at 25 MHz. All using standard electrically erasable CMOS GAL technology.

These remarkable figures are a result of the MAPL's patented paged architecture, which divides the 128 product terms into 8 pages. This paging is invisible to the user, as software fitters automatically assign state bits to enable pages. There is no timing penalty for switching pages. The architecture limits each state to a maximum of 16 product terms, but due to the optimized macrocell design, this limitation is rarely encountered. Paging powers up only the portion of the array needed by the current clock cycle. Since the array sense amplifiers can be shared among the pages, power can be devoted to device speed instead of device size. The benefit to the user is a PLA that provides 128 product terms to all outputs at 45 MHz *(Figure 5)*. The MAPL is an architectural solution to state machine applications.

The MAPL PLA drives 27 registered macrocells, with two sum terms for each cell *(Figure 6)*. These macrocells have been designed to enhance the overall device functionality

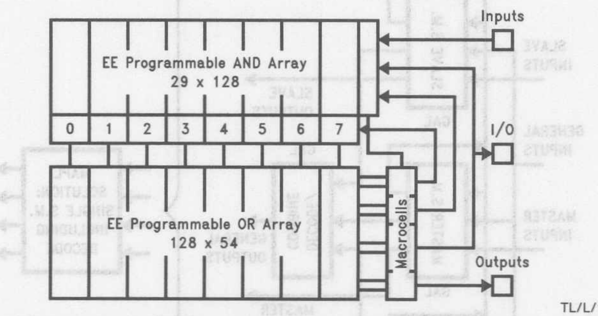


FIGURE 5. Functionally, the MAPL appears to be a continuous 128-product term PLA.

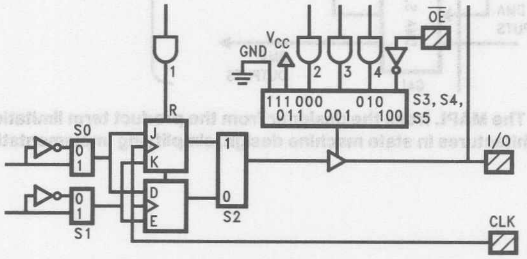


FIGURE 6. The MAPL two-input macrocell includes programmable polarity and DE and JK flip-flops to allow selection of the optimum function of any application.

ther DE (D with clock enable) or JK. This allows selection of the default (no product term active) condition of the output to be either set, reset, hold or toggle. This can have a significant impact on the number of product terms used in any state, depending on the type of structure present in the design. Additional PAL product terms are used to provide asynchronous register reset and a selection of four output enable functions to all signals.

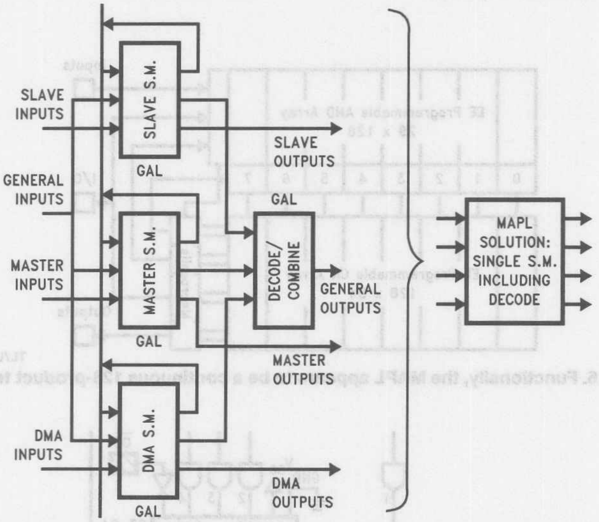
Note that the array is fully registered, that is, no combinatorial outputs are available. However, for synchronous state machine applications, this is not a problem. In most designs, combinatorial and asynchronous functions are added to the periphery of the core state machine.

STATE MACHINE DESIGN TECHNIQUES

The low performance of traditional PLA architectures have forced most high-speed state machine designs to be implemented in PAL architecture devices. This requires designers to compromise their ideal design methodology, due to the limitations of the PAL architecture in these applications.

PALs and GALs are ideal for very simple state machines. They have short propagation delays, and these delays are uct terms available to each output begins to squeeze the design. This problem occurs even in the new high-density partitioned PAL architectures. Solutions include breaking a single complex state machine into several smaller ones, or encoding inputs and decoding outputs in separate devices. The result is a design with worst-case delays through multiple devices, and signals with a wide range of timing values. Worse yet, because of the contortions necessary to fit the design into the product terms available, design modifications or enhancements are difficult to impossible. Equation-level entry and a detailed knowledge of the device architecture are often required. Rather than providing a solution, PAL state machine design becomes a challenge.

An example of this is a typical bus interface *(Figure 7)*. Although the slave, master, and DMA bus operations logically form a single, large state machine, PAL product term limitations force the designer to break them up into three separate interconnected state machines. Some inputs and outputs are independent, but others are common to the entire design, and must be shared among mutiple inputs and outputs. The resulting network of feedbacks and decodes result in a complex design with inconsistent timing.



TL/L/11296-7

FIGURE 7. The MAPL frees the designer from the product term limitations of PAL architectures in state machine design, simplifying implementation.

A PLA architecture, in contrast, provides ample product terms to the user. This allows complex state machines to be implemented in a single device. All outputs have identical timing, and adding a product term or a state is no longer a major modification. Best of all, it is possible to design in software at the state machine level, with all the benefits that has for design maintenance and documentation.

The MAPL is the first device to solve the PLA performance problem without compromising functionality. A closer look at the characteristics of state machines themselves explains why.

A state machine, by definition, has only one state active at a time. During the clock cycle in which that state is active, only the equations necessary to determine the next state and outputs are being used. Out of the entire design, only the product terms needed by one state can be active. In a traditional PLA, sufficient power must be provided to drive all product terms, regardless of how many are used by the current state. The MAPL, by paging product terms, drives only 16 terms per clock cycle. It is extremely unusual for a single state to require more than 16 product terms. In fact, several states will usually fit onto a single page. This means that paging can be fully transparent to the user. The full state machine has access to 128 product terms, while the device only needs to provide sufficient current to drive 16 product terms. As a result, the MAPL can be both big and fast.

The MAPL two-input macrocell also serves to increase the device's functionality. A signal which is normally set or reset, and only asserted in one or two states, is well-suited to a standard D flip-flop with polarity selection. A signal that changes state only occasionally, but is active over a large range of states, would require product terms in every one of these states. By using the MAPL's DE flip-flop, one product term can be used to turn the output on, and another to turn it off. Holding the output active uses no product terms. Counters can also require a large number of product terms when implemented in D flip-flops. With the MAPL's JK flip-flop, only a single term is needed for each bit of the counter. A 16-bit counter fits on a single page of the MAPL, leaving the remaining seven pages available for the rest of a state machine. Macrocell type selection decreases product term usage and increases the effective density of the MAPL.

In the bus interface example above, the MAPL would be able to implement the interface state machine directly, resulting in a simpler design and fewer chips, with consistent timing and no loss of system speed.

THE MAPL FAMILY

National is developing two families of MAPL devices. The first, the MAPL1 family, is focused only on the state machine portion of a design. The MAPL128, in production, is a 28-pin device incorporating the 128-product term MAPL paged array described above. It has 9 inputs, 4 dedicated outputs, 12 I/O, 8 buried feedbacks, and 3 page bits which can be used as state feedbacks. One of the inputs can also be used as a fast output enable, along with the three dedicated product terms. All 27 macrocells can be asynchronously reset. The MAPL144 is a 44-pin device based on the same internal architecture, but the 8 buried feedbacks are also available as outputs. All run at a system clock rate of 45 MHz.

The MAPL2 family brings in additional functions that are needed in some applications. It is not uncommon for a state machine to require that its inputs or outputs be combined with other high-speed signals, sometimes with specialized timing. The MAPL2 includes 8 PAL outputs with dynamic polarity control which can be used in conjunction with or independently of the MAPL array. Either registered or combinatorial outputs are available. The MAPL2 also has optional input latches on all signals to reduce setup times on critical signals, and double-buffering for synchronizing asynchronous inputs to avoid metastability.

TABLE I. MAPL Family

MAPL Product	128	144	244	268
Package Size	28	44	44	68
Max Inputs	21	21	32	36
Max Outputs	16	24	32	40
Macrocells	27	27	35	43
Total I/O	25	33	32	24
System Speed	40 MHz	45 MHz	50 MHz	50 MHz
PLA Product Terms	128	128	128	128
PAL Product Terms	4	4	76	104
Max Power at 25 MHz	110 mA	125 mA	180 mA	

PLD DESIGN SOFTWARE

In order to design effectively with any PLD, suitable design tools must be used. Because a designer's time is limited, it is important that they require little or no additional effort to learn. For the same reason, they must provide entry formats which suit the application at hand.

In the ideal world, a designer would only need to learn a single design environment. It takes time for new devices to be supported across all platforms, however, so there is a place for manufacturers to provide their own low-cost software to bridge the gap. National Semiconductor, for example, sells OPAL™ as a complete design package for PAL, GAL, and MAPL users. It provides boolean equation, truth table, and state machine language entry formats, and generates JEDEC maps and functional simulations for all National PLDs. A menu-driven shell environment makes it easy to use as a stand-alone tool, and the MAPL fitter automatically minimizes and compiles designs into the MAPL paged architecture.

For those that need to integrate the latest in PLDs with their own tools, it is important to provide a bridge between software packages. National is one of the vendors supporting the use of Berkeley's Espresso PLA format. Rapidly becoming a standard for the transfer of logic design, it can be read or generated by a wide range of design tools. Combined with Data I/O's OPEN PLA device-specific "dot extensions," proprietary fitters, such as for the MAPL, can be used with third-party software. Indeed, National provides the MAPL fitter to all PLD software vendors, making it easy for them to provide the latest support quickly and easily.

Having broad software support for a product is essential to users, who need to be able to evaluate a wide range of devices for a particular application quickly. If a single state machine can be compiled into four or five devices in a few minutes, a real evaluation of performance and functionality can take place. Therefore, manufacturer support for an OPEN PLA format provides real benefit to both users, who gain access to technology quickly, and to manufacturers, who present the lowest possible hurdle to use of their products.

STATE MACHINE DESIGN EXAMPLES

A traffic light controller is a classic state machine design example. Here it is used to demonstrate characteristics of state machine language design and the Berkeley PLA format. National's OPAL syntax is used to describe the state machine, but the function should be fairly obvious to anyone with state machine language experience.

The version of the controller shown below has three inputs, aside from the clock. The "clr" signal is used to asynchronously reset the state machine, and the two "sensor" signals detect the presence of cars at the intersection. The size outputs of the design are the green, yellow, and red lights for each direction. Notice that the characteristics of each output have been taken into account in the MAPL macrocell selection. Both use a DE flip-flop, but the red and green lights, which are on or off throughout several states, hold their state as the default condition, while the yellow light, normally off, uses the reset state as its normal condition. Polarity selection at the input to the macrocells physically implements these defaults.

The state bits themselves are defined as JK flip-flops, with a default of toggle. This is because the state machine has a 2-bit counter embedded in it to control the maximum length of time any car can be kept waiting. As JK flip-flops implement efficient counters, this will minimize product term usage. These PLA outputs are buried within the MAPL, as they are not needed by any external logic. The "stateReg" set of symbols is defined in order that the asynchronous state reset function can be defined in a single equation below. The state values are defined in order to ensure that the 2-bit counters (states A1–A4 and B1–B4) follow a linear sequence. The remainder of the assignments, except for the Start state, are of little consequence.

The equation block assigns the "clr" input to the asynchronous reset product term of the state bits. The other equations are merely definitions for variables used within the rest of the state machine.

The state machine consists of 13 states, two of which have branch conditions. The green and red lights must have each of their transitions explicitly defined, while the yellow lights only need to have their active conditions defined. The undefine state is reserved within the OPAL syntax. It is used to guarantee that the state machine returns to the Start state if any of the 3 undefined state bit encodings are encountered.

```
BEGIN HEADER
   OPAL Traffic Signal Controller, D. Hawley, NSC, 15 Aug 1991
END HEADER

BEGIN DEFINITION
   device  MAPL128;
   inputs  clk, clr, sensorA, sensorB;
   outputs (de, hold) greenA, redA, greenB, redB;
   outputs (de, rst) yellowA, yellowB;
   statebits (buried, jk, toggle) s3, s2, s1, s0;
   set stateReg = [s3,s2,s1,s0];
   state_names Start=0, A0=3, A1=4, A2=5, A3=6, A4=7, A5=8,
            B0=9, B1=12, B2=13, B3=14, B4=15, B5=2;
END DEFINITION
```

TL/L/11296–11

```
BEGIN EQUATION
   stateReg.ar=clr;
   turnedOn=1;
   turnedOff=0;
   On=1;
END EQUATION

BEGIN STATE_DIAGRAM
   state Start: goto A0 with
         greenA:=turnedOff; yellowA:=On; redA:=turnedOn;
         greenB:=turnedOff; redB:=turnedOn;
         endwith;
   state A0: goto A1 with
         greenA:=turnedOn; redA:=turnedOff;
         endwith;
   state A1: if ( sensorA & !sensorB ) then A1
      else if (!sensorA &  sensorB ) then A5 with
         greenA:=turnedOff; yellowA:=On;
         endwith
      else if ( sensorA == sensorB ) then A2;
   state A2: goto A3;
   state A3: goto A4;
   state A4: goto A5 with
         greenA:=turnedOff; yellowA:=On;
         endwith;
   state A5: goto B0 with
         redA:=turnedOn;
         yellowB:=On;
         endwith;
   state B0: goto B1 with
         greenB:=turnedOn; redB:=turnedOff;
         endwith;
   state B1: if (!sensorA &  sensorB ) then B1
      else if ( sensorA & !sensorB ) then B5 with
         greenB:=turnedOff; yellowB:=On;
         endwith
      else if ( sensorA == sensorB ) then B2;
   state B2: goto B3;
   state B3: goto B4;
   state B4: goto B5 with
         greenB:=turnedOff; yellowB:=On;
         endwith;
   state B5: goto A0 with
         redB:=turnedOn;
         yellowA:=On;
         endwith;
   state undefine: goto Start;
END STATE_DIAGRAM
```

TL/L/11296-9

since the state bits can be buried, more I/O are free for additional functions as well. Even in this basic example, the advantages of the MAPL architecture for state machine design is evident.

The entire traffic signal controller can also be defined in the Berkeley PLA format shown below. This OPEN PLA format allows the same design to be transferred from one design tool package to another. National's OPAL software compiled the state machine above into a Berkeley PLA file, performed the minimization and page assignment required by the MAPL, and generated the revised PLA file shown below. This output can then be compiled into a JEDEC map for programming the MAPL128.

The PLA file format is extremely compact. The lines beginning with "#$" are comments used by National's software ed the two macrocell inputs required by the MAPL DE or JK flip-flops, and defined the necessary polarity selections with the "-" suffix. The ".type" and ".phase" definitions are not used by the OPAL software. The ".p" line shows the total number of product terms in the design, including the 4 identical asynchronous reset terms at the bottom. The PLA map itself is in truth-table format, with each column defining an input or output in the order defined above. Each product term is represented by a single row. A "-" defines a "don't care" input, while "~" defines the same for outputs.

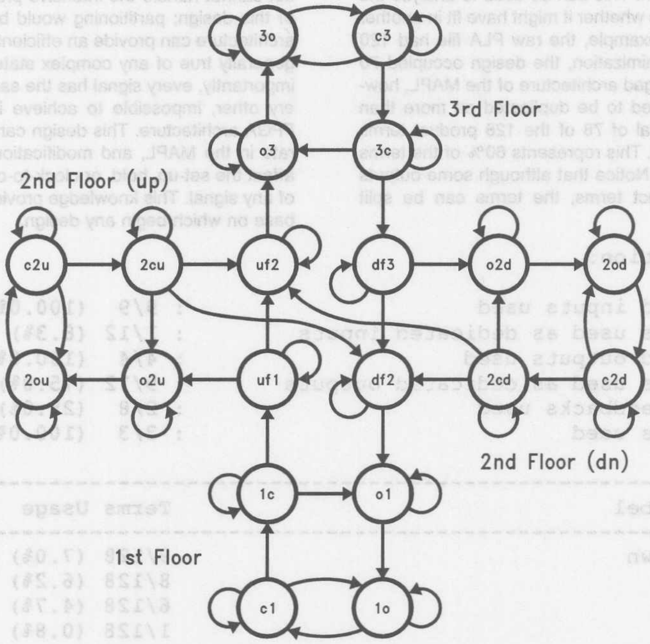
This is the format which allows software from different vendors to interchange PLD design information. Any software package which generates this file can be used as the design entry vehicle for the MAPL device. A PLD design tool with specific MAPL support, such as OPAL, can then perform the JEDEC map creation required for programming.

```
#$ TOOL NSC
#$ TITLE OPAL Traffic Signal Controller, D. Hawley, NSC, 15 Aug 1991
#$ TITLE
#$ DEVICE MAPL128
.i 10
.o 26
#$ PINS  10 clk:6 clr:5 sensorA:4 sensorB:3 greenA:12 redA:13 greenB:15
redB:16 yellowA:11 yellowB:10
#$ NODES 6 s3:37 s2:30 s1:31 s0:32 pb2:39 pb1:38
.ilb clk clr sensorA sensorB s3 s2 s1 s0 pb2 pb1
.ob  s3.j s3.k s2.j s2.k s1.j s1.k s0.j s0.k greenA.d greenA.ce redA.d
redA.ce greenB.d greenB.ce redB.d redB.ce yellowA.d yellowA.ce-
yellowB.d yellowB.ce- s0.ar s1.ar s2.ar s3.ar pb2.reg pb1.reg
.type f
.phase 11111111111111111111111111
.p 18
--01 010000 11110000 010000001000000000
---- 100000 00000000 001100000010000000
--10 110000 11111100 000001000010000000
--00 --0-00 00000011 000000000000000000
--11 --0-00 00000011 000000000000000000
---- 100100 00110000 000011010000000000
---- 111100 11110000 000001000010000000
---- 000000 00001100 011101000000000000
---- 00-000 00000011 000000111000000000
---- -11-00 00000011 000000000000000000
---- -00-00 00000011 000000000000000000
---- 011100 11111100 010000001000000000
---- -10100 00001111 000000000000000000
---- 001100 00111111 110100000000000000
-1-- ------ ~~~~~~~~ ~~~~~~~~~~~~~~~1~~
-1-- ------ ~~~~~~~~ ~~~~~~~~~~~~~~1~~~
-1-- ------ ~~~~~~~~ ~~~~~~~~~~~~~1~~~~
-1-- ------ ~~~~~~~~ ~~~~~~~~~~~~1~~~~~
.e
```

TL/L/11296-10

lift. It has 10 inputs, which include a call button for each floor, an "open" button inside the lift, and sensor inputs

created the 5 state bits automatically. The full state diagram is too long to present, but a sample is shown below.



TL/L/11296-8

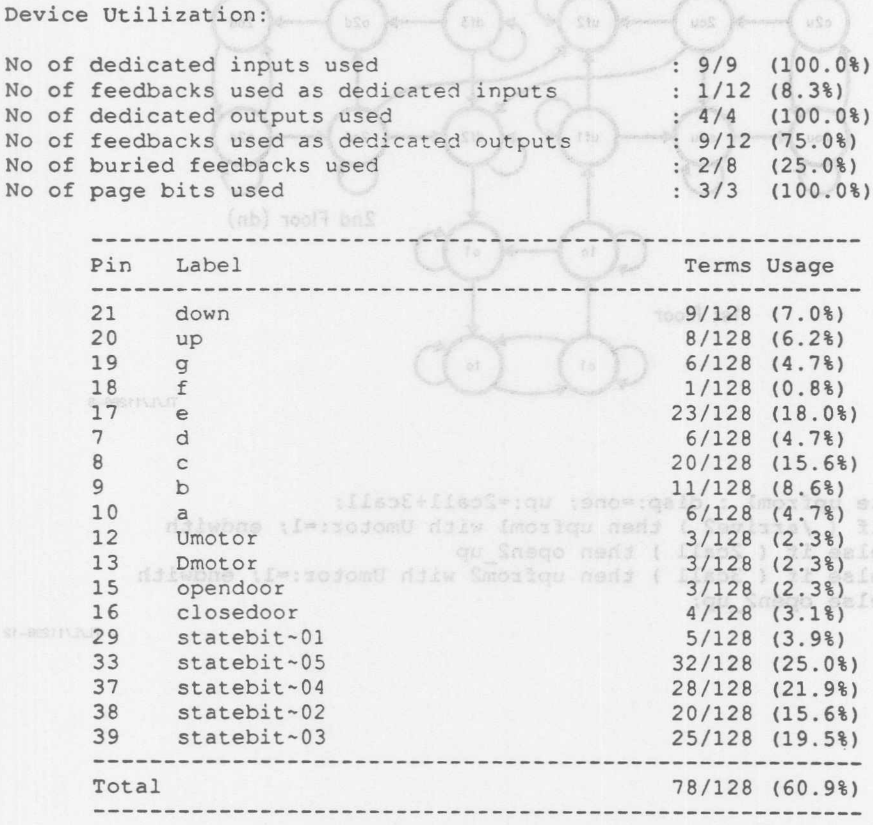
```
state upfrom1 : disp:=one; up:=2call+3call;
    if ( /arrive2 ) then upfrom1 with Umotor:=1; endwith
    else if ( 2call ) then open2_up
    else if ( 3call ) then upfrom2 with Umotor:=1; endwith
    else open2_up;
```

TL/L/11296-12

In this state example (uf1), the display decode is specified in the first line. An if-then-else syntax is used to describe the branch conditions, and the up motor is driven if the lift has not yet arrived at its destination. The remaining states are of similar complexity.

The OPAL software generates a log file which summarizes the total device utilization. This can be used to analyze the design, and to determine whether it might have fit in another type of device. In this example, the raw PLA file had 120 product terms. After minimization, the design occupied 70 product terms. In the paged architecture of the MAPL, however, a few terms needed to be duplicated on more than one page, so a final total of 78 of the 128 product terms were actually consumed. This represents 60% of the terms available on the device. Notice that although some outputs require up to 32 product terms, the terms can be split among multiple pages because only a few are used in each state. The sum of the individual output terms is less than the total because terms can be shared among multiple outputs in a PLA architecture.

No PAL could implement this state machine in a single device. Even the highest-density PAL architecture on the market cannot handle the intensive product term requirements of this design; partitioning would be required. Only a PLA architecture can provide an efficient implementation. This is generally true of any complex state machine design. Most importantly, every signal has the same timing relative to every other, impossible to achieve in a partitioned PAL or FPGA architecture. This design can run at a 45 MHz clock rate in the MAPL, and modification of the design will not affect the set-up, hold, or clock-to-output timing parameters of any signal. This knowledge provides the user with a solid base on which begin any design.



```
Device Utilization:

No of dedicated inputs used                      : 9/9   (100.0%)
No of feedbacks used as dedicated inputs         : 1/12  (8.3%)
No of dedicated outputs used                     : 4/4   (100.0%)
No of feedbacks used as dedicated outputs        : 9/12  (75.0%)
No of buried feedbacks used                      : 2/8   (25.0%)
No of page bits used                             : 3/3   (100.0%)

          ------------------------------------------------------
          Pin   Label                          Terms Usage
          ------------------------------------------------------
          21    down                           9/128 (7.0%)
          20    up                             8/128 (6.2%)
          19    g                              6/128 (4.7%)
          18    f                              1/128 (0.8%)
          17    e                             23/128 (18.0%)
          7     d                              6/128 (4.7%)
          8     c                             20/128 (15.6%)
          9     b                             11/128 (8.6%)
          10    a                              6/128 (4.7%)
          12    Umotor                         3/128 (2.3%)
          13    Dmotor                         3/128 (2.3%)
          15    opendoor                       3/128 (2.3%)
          16    closedoor                      4/128 (3.1%)
          29    statebit~01                    5/128 (3.9%)
          33    statebit~05                   32/128 (25.0%)
          37    statebit~04                   28/128 (21.9%)
          38    statebit~02                   20/128 (15.6%)
          39    statebit~03                   25/128 (19.5%)
          ------------------------------------------------------
          Total                               78/128 (60.9%)
          ------------------------------------------------------
```

TL/L/11296–13

SUMMARY

PAL architectures are ideal for high performance logic replacement. The new generation of high-density PAL architectures provide greater integration for applications with a limited number of product terms. State machine applications require a large number of shared product terms, and are best suited to PLA architectures. National's MAPL128 is the first of a new family of paged-PLA devices which provide high performance for complex designs. FPGAs provide high levels of integration, but performance is extremely variable due to interconnect delays.

The best way to evaluate a technology is to compile a design onto a range of devices. Manufacturer support of open PLA formats guarantees that users have access to the latest devices within a consistent design framework. National Semiconductor provides access to MAPL devices through its own OPAL software and by working closely with PLD software vendors worldwide. In this way, state machines can be integrated as a part of a board design.

A Smart UART Design Implemented in a MAPL128

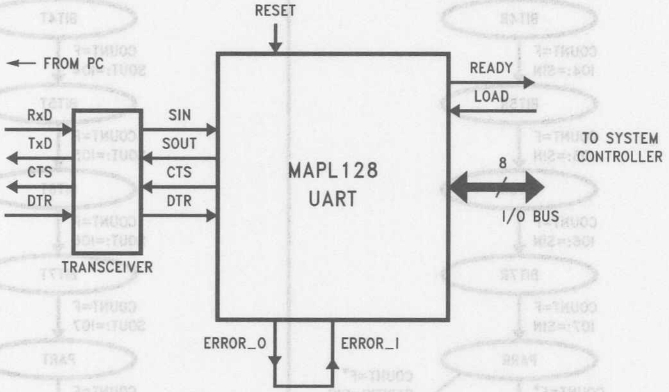
National Semiconductor
Application Note 800
David O'Neal

A "smart" UART is often necessary to interface to a PC when designing a stand-alone circuit. This application note describes a UART design implemented in a MAPL128 programmable logic device. The state diagram, sample timing diagram, and OPAL™ entry language listing are included. This UART design utilizes 77% of the MAPL128 device, allowing the designer the flexibility of changing or adding to the design.

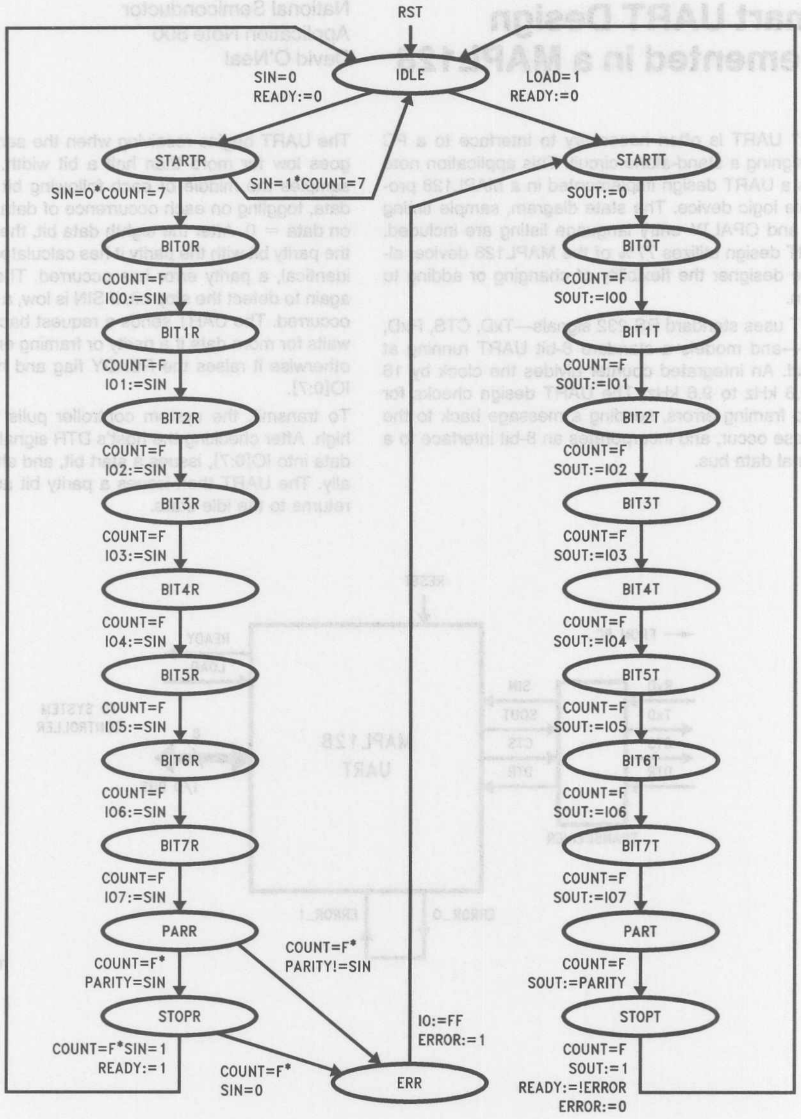
The UART uses standard RS-232 signals—TxD, CTS, RxD, and DTR—and models a standard 8-bit UART running at 9600 baud. An integrated counter divides the clock by 16 from 153.6 kHz to 9.6 kHz. The UART design checks for parity and framing errors, sending a message back to the host if these occur, and incorporates an 8-bit interface to a bidirectional data bus.

The UART begins receiving when the serial in (SIN) signal goes low for more than half a bit width. The UART then samples the middle of each following bit to ensure stable data, toggling on each occurrence of data = 1 and holding on data = 0. After the eighth data bit, the UART compares the parity bit with the parity it has calculated. If these are not identical, a parity error has occurred. The SIN is sampled again to detect the stop bit. If SIN is low, a framing error has occurred. The UART sends a request back to the host and waits for more data if a parity or framing error has occurred, otherwise it raises the READY flag and holds the data on IO[0:7].

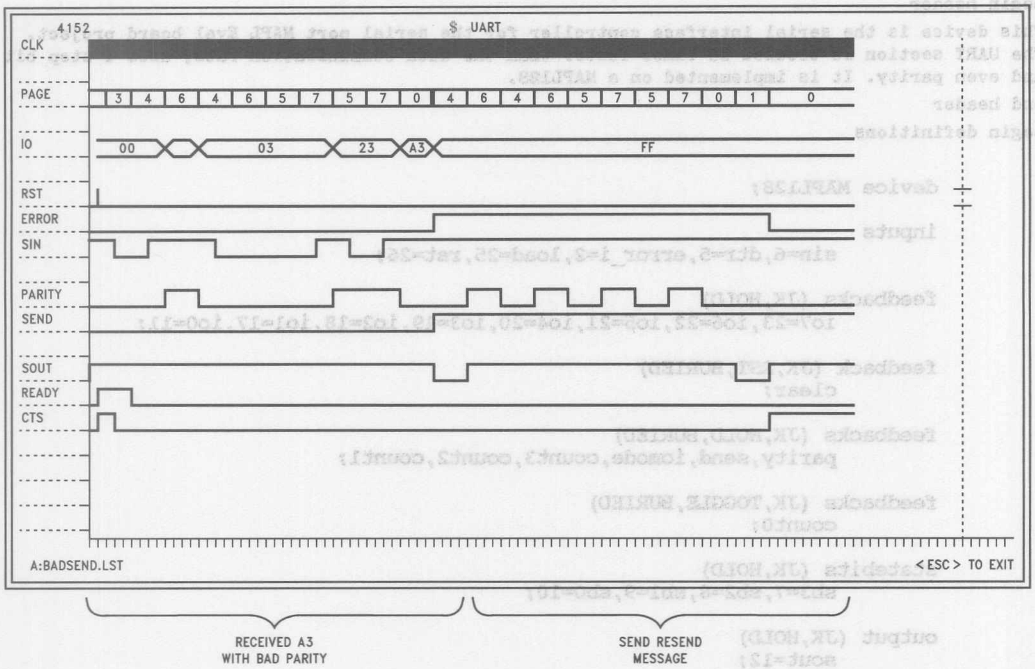
To transmit, the system controller pulls the LOAD signal high. After checking the host's DTR signal, the UART loads data into IO[0:7], issues a start bit, and shifts the data serially. The UART then issues a parity bit and a stop bit and returns to the idle state.



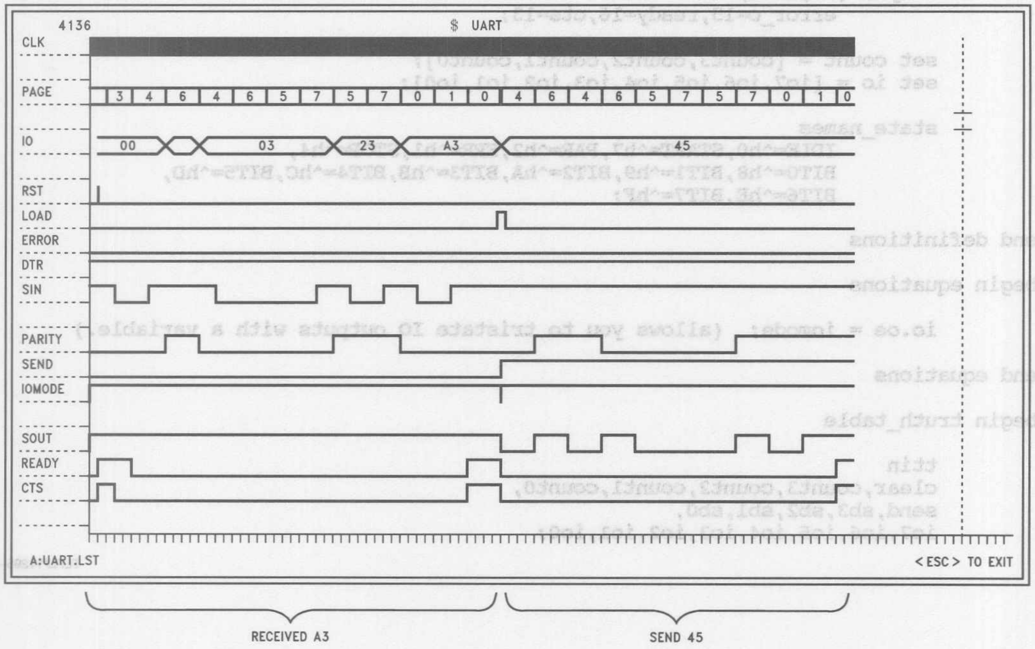
TL/L/11295-1


RST
IDLE
SIN=0
READY:=0
LOAD=1
READY:=0
STARTR
SIN=0*COUNT=7
SIN=1*COUNT=7
BIT0R
COUNT=F
IO0:=SIN
BIT1R
COUNT=F
IO1:=SIN
BIT2R
COUNT=F
IO2:=SIN
BIT3R
COUNT=F
IO3:=SIN
BIT4R
COUNT=F
IO4:=SIN
BIT5R
COUNT=F
IO5:=SIN
BIT6R
COUNT=F
IO6:=SIN
BIT7R
COUNT=F
IO7:=SIN
PARR
COUNT=F*
PARITY=SIN
COUNT=F*
PARITY!=SIN
STOPR
COUNT=F*SIN=1
READY:=1
COUNT=F*
SIN=0
ERR
IO:=FF
ERROR:=1
STARTT
SOUT:=0
BIT0T
COUNT=F
SOUT:=IO0
BIT1T
COUNT=F
SOUT:=IO1
BIT2T
COUNT=F
SOUT:=IO2
BIT3T
COUNT=F
SOUT:=IO3
BIT4T
COUNT=F
SOUT:=IO4
BIT5T
COUNT=F
SOUT:=IO5
BIT6T
COUNT=F
SOUT:=IO6
BIT7T
COUNT=F
SOUT:=IO7
PART
COUNT=F
SOUT:=PARITY
STOPT
COUNT=F
SOUT:=1
READY:=!ERROR
ERROR:=0

TL/L/11295-2



TL/L/11295-3



TL/L/11295-4

```
begin header
This device is the serial interface controller for the serial port MAPL Eval board project.
The UART section is clocked 16 times faster than the data communication rate, uses 1 stop bit
and even parity. It is implemented on a MAPL128.
end header
begin definitions

        device MAPL128;

        inputs
                sin=6,dtr=5,error_i=2,load=25,rst=26;

        feedbacks (JK,HOLD)
                io7=23,io6=22,io5=21,io4=20,io3=19,io2=18,io1=17,io0=11;

        feedback (JK,RST,BURIED)
                clear;

        feedbacks (JK,HOLD,BURIED)
                parity,send,iomode,count3,count2,count1;

        feedbacks (JK,TOGGLE,BURIED)
                count0;

        statebits (JK,HOLD)
                sb3=7,sb2=8,sb1=9,sb0=10;

        output (JK,HOLD)
                sout=12;

        output (JK,HOLD)
                error_o=15,ready=16,cts=13;

        set count = [count3,count2,count1,count0];
        set io = [io7,io6,io5,io4,io3,io2,io1,io0];

        state_names
                IDLE=^h0,START=^h7,PAR=^h2,ERR=^h1,STOP=^h4,
                BIT0=^h8,BIT1=^h9,BIT2=^hA,BIT3=^hB,BIT4=^hC,BIT5=^hD,
                BIT6=^hE,BIT7=^hF;

end definitions

begin equations

        io.oe = iomode;  {allows you to tristate IO outputs with a variable.}

end equations

begin truth_table

        ttin
        clear,count3,count2,count1,count0,
        send,sb3,sb2,sb1,sb0,
        io7,io6,io5,io4,io3,io2,io1,io0;
```

TL/L/11295-5

```
        ttout
        count3,count2,count1,count0,
        sb3,sb2,sb1,sb0,
        io7,io6,io5,io4,io3,io2,io1,io0;

        1---- - ---- --------   0001 ---- -------- {counter used as a clock  }
        0---1 - ---- --------   --!- ---- -------- {divider.  It resets to   }
        0--11 - ---- --------   -!-- ---- -------- {0001 to account for the  }
        0-111 - ---- --------   !--- ---- --------{one clock needed to reset}

        ----- 1 0111 1-------   ---- 1000 1-------{this part of the truth }
        ----- 1 0111 -1------   ---- 1000 -1------{table loads the IO      }
        ----- 1 0111 --1-----   ---- 1000 --1-----{variables using only 8 }
        ----- 1 0111 ---1----   ---- 1000 ---1---- {product terms (since in}
        ----- 1 0111 ----1---   ---- 1000 ----1---{the previous state they}
        ----- 1 0111 -----1--   ---- 1000 -----1--{were all set to zero). }
        ----- 1 0111 ------1-   ---- 1000 ------1-{Normally with T ff's it}
        ----- 1 0111 -------1   ---- 1000 -------1 {would take 16 product  }
                                                   {terms.                 }

end truth_table

begin state_diagram

        state ALL :                                {set the reset conditions}
                if rst then IDLE with
                        send := 0;
                        iomode := 1;
                        error_o := 0;
                        sout := 1;
                        ready := 1;
                        cts := 1; endwith;

        state IDLE :
                if /sin then START with             {detect a start bit}
                        clear := 1;
                        send := 0;
                        cts := 0; endwith
                else if load*dtr then START with    {begin a send cycle}
                        send := 1;
                        iomode := 0;
                        io := ^h00;
                        cts := 0;
                        ready := 0; endwith
                else IDLE with iomode := 1;
                               sout := 1; endwith;

        state START :
                case
{waiting to verify}     /send*(count != ^h8) : START;
{start bit}

{start bit verified}    /send*(count == ^h8)*/sin*/clear : BIT0 with
                                clear := 1;
                                ready := 0;
                                parity := 0; endwith;

{false start bit }      /send*(count == ^h8)*sin*/clear : IDLE with
{return to IDLE}                cts := 1; endwith;
```

TL/L/11295-6

```
{send out start bit}  send : BIT0 with
                             clear := 1;
                             parity := 0;
                             sout := 0;
                             iomode := 1; endwith;
               endcase;

       state BIT0 :                             {states BIT0 - BIT7 are}
               if count!=^hf then BIT0          {identical.  Wait until}
               else if /send then BIT1 with     {the counter reaches 15}
                      io0 := sin;               {then receive input or }
                      parity.j = sin;           {send output. In either}
                      parity.k = sin; endwith   {case, toggle parity   }
               else BIT1 with                   {bit if a one is sent  }
                      sout := io0;              {or received.          }
                      parity.j = io0;
                      parity.k = io0; endwith;

       state BIT1 :
               if count!=^hf then BIT1
               else if /send then BIT2 with
                      io1 := sin;
                      parity.j = sin;
                      parity.k = sin; endwith
               else BIT2 with
                      sout := io1;
                      parity.j = io1;
                      parity.k = io1; endwith;

       state BIT2 :
               if count!=^hf then BIT2
               else if /send then BIT3 with
                      io2 := sin;
                      parity.j = sin;
                      parity.k = sin; endwith
               else BIT3 with
                      sout := io2;
                      parity.j = io2;
                      parity.k = io2; endwith;

       state BIT3 :
               if count!=^hf then BIT3
               else if /send then BIT4 with
                      io3 := sin;
                      parity.j = sin;
                      parity.k = sin; endwith
               else BIT4 with
                      sout := io3;
                      parity.j = io3;
                      parity.k = io3; endwith;

       state BIT4 :
               if count!=^hf then BIT4
               else if /send then BIT5 with
                      io4 := sin;
                      parity.j = sin;
                      parity.k = sin; endwith
               else BIT5 with
                      sout := io4;
                      parity.j = io4;
```

TL/L/11295-7

```
                        parity.k = io4; endwith;

        state BIT5 :
                if count!=^hf then BIT5
                else if /send then BIT6 with
                        io5 := sin;
                        parity.j = sin;
                        parity.k = sin; endwith
                else BIT6 with
                        sout := io5;
                        parity.j = io5;
                        parity.k = io5; endwith;

        state BIT6 :
                if count!=^hf then BIT6
                else if /send then BIT7 with
                        io6 := sin;
                        parity.j = sin;
                        parity.k = sin; endwith
                else BIT7 with
                        sout := io6;
                        parity.j = io6;
                        parity.k = io6; endwith;

        state BIT7 :
                if count!=^hf then BIT7
                else if /send then PAR with
                        io7 := sin;
                        parity.j = sin;
                        parity.k = sin; endwith
                else PAR with
                        sout := io7;
                        parity.j = io7;
                        parity.k = io7; endwith;

        state PAR :
                case
                        count != ^hf : PAR;

{error if parity}       (count == ^hf)*/send*(sin != parity) : ERR with
{does not match }               error_o := 1; endwith;

{continue if it does}   (count == ^hf)*/send*(sin == parity) : STOP;

{output parity on send} (count == ^hf)*send : STOP with
                                iomode := 1;
                                sout := parity; endwith;
                endcase;


        state STOP :
                case
                        count != ^hf : STOP;

{error if no stop}      (count == ^hf)*/send*/sin : ERR with
{bit is detected }              error_o := 1; endwith;

{return to IDLE if  }   (count == ^hf)*/send*sin : IDLE with
{there is a stop bit}           cts := 1;
                                ready := 1; endwith;
```

TL/L/11295-8

```
{return to IDLE with}   (count == ^hf)*send*/error_i : IDLE with
{ready high if this }           cts := 1;
{was a normal send  }           sout := 1;
                                ready := 1; endwith;

{return to IDLE with }  (count == ^hf)*send*error_i : IDLE with
{ready low if this   }          cts := 1;
{was an error message}          sout := 1;
{being sent          }          error_o := 0; endwith;

                endcase;

        state ERR :
                if dtr then START with         {load error message (FF) in}
                        iomode := 1;           {IO registers and send it  }
                        io := ^hff;            {to the host.              }
                        send := 1; endwith
                else ERR;

end state_diagram
```

TL/L/11295-9

State Machine Design

National Semiconductor
Application Note 801
Bill Carlson, Snr. Field
Applications Engineer

Multiple Array Programmable Logic, or MAPL™ is National Semiconductor's new family of high density programmable logic devices. These PLDs are ideally suited for state machines, controllers, and synchronous logic applications. The purpose of this application note is to show how easy it is to implement state machines in MAPL devices. It is assumed that the reader is familiar with the MAPL128 data sheet, general principles of logic design, and has knowledge of PLD design.

- State machine fundamentals
- The state machine design process
- State machine design using MAPL and OPAL™
- The MAPL state machine architecture advantage

STATE MACHINE FUNDAMENTALS

Implementing a complex state machine with a MAPL device is much easier than with conventional programmable logic. To facilitate a MAPL implementation, an understanding of state machine design principles may be needed. This section and the next will review these principles. Once a design example is illustrated, the capabilities of MAPL will become more apparent.

State machines are used in nearly all digital systems. State machines often resolve timing dissimilarities between different devices and enable them to work together. They also perform the functions of complex counters and micro-instruction sequencers. A state machine is needed when a microprocessor interfaces to another large VLSI device, such as a storage controller. State machines are also needed when an adapter board needs to gain access to the system bus, or when a CPU needs to interface to a complex memory architecture. In a slave mode or bus master mode transfer, state machines are used to translate the signals of a bus to those of a device the bus is accessing. In bus masters, state machines implement bus arbitration, the bus transfer protocol, and limiting bus tenure times upon being pre-empted. There are also serial protocols that need a complex state machine to do frame and address recognition, frame stripping, encoding/decoding and controlling the serializing and de-serializing of data.

State machines are based on sequential logic design where the inputs of the circuit along with the previous outputs determine what the next outputs will be. There are two different types of state machines, the Mealy and the Moore models as shown in *Figure 1*.

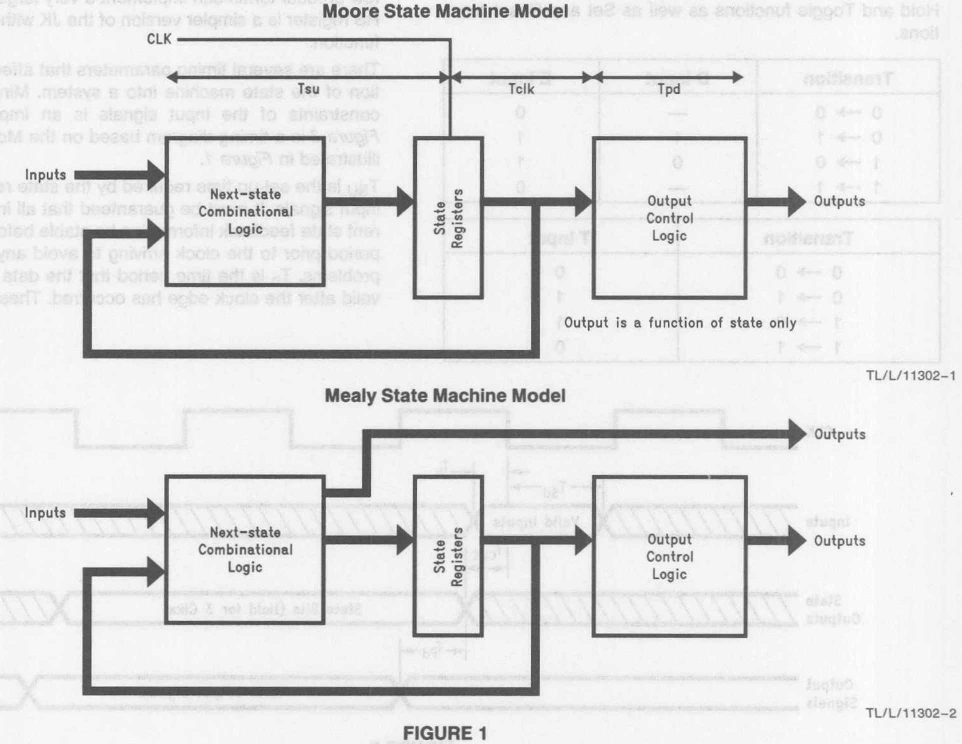

Moore State Machine Model
CLK
Tsu
Tclk
Tpd
Inputs
Next-state Combinational Logic
State Registers
Output Control Logic
Outputs
Output is a function of state only
TL/L/11302-1
Mealy State Machine Model
Outputs
Inputs
Next-state Combinational Logic
State Registers
Output Control Logic
Outputs
TL/L/11302-2

FIGURE 1

In the Moore model, the outputs are only dependent upon the current state. In the Mealy model, by contrast, the outputs are dependent not only on the present state, but also on the present inputs. Once a certain state is entered, the state bits are fed back to the combinational logic where output functions change asynchronously with respect to the clock. When an input changes, a corresponding output will change as well, even between clock periods. This is in contrast to the Moore model where an input change won't affect the output until the next clock cycle has occurred and a new state is entered. This application note will focus on the more common Moore type.

A state is defined as the current value of a group of state registers; each individual register value is called a state bit. The outputs are decoded from the current state and are used to control the actual functions in the system. The next-state logic is combinatorial in nature and combines the current state information with the inputs to form the logic needed to determine the next state. This logic can be represented by equations defining the inputs to the flip-flops in the state registers. These equations are dependent upon the register used, whether D/E, J/K, R/S, or T. The type of register chosen affects the complexity of this logic.

Each type of register has its advantages and disadvantages. In order to derive the equations for transitioning a register from one state to another, a transition table is needed for the type of register being implemented. These tables are somewhat different than the flip-flop truth tables as they are usually defined. The register transition tables define transitions only and assume a known current state to a known next state. Flip-flop truth tables on the other hand define Hold and Toggle functions as well as Set and Reset functions.

Transition	D Input	E Input
0 → 0	—	0
0 → 1	1	1
1 → 0	0	1
1 → 1	—	0

Transition	T Input
0 → 0	0
0 → 1	1
1 → 0	1
1 → 1	0

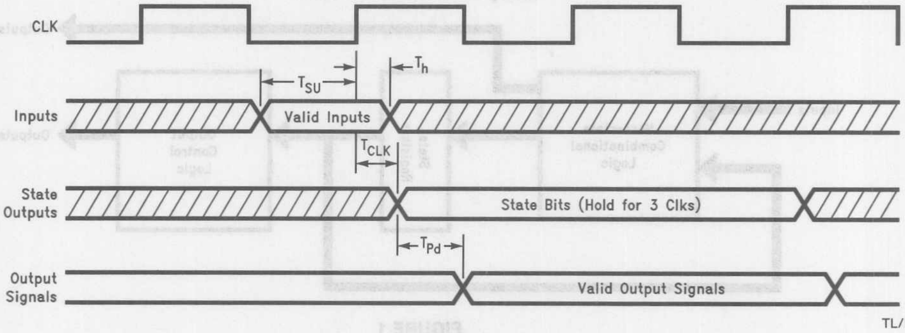
Transition	J Input	K Input
0 → 0	0	—
0 → 1	1	—
1 → 0	—	1
1 → 1	—	0

Transition	S Input	R Input
0 → 0	0	—
0 → 1	1	—
1 → 0	—	1
1 → 1	—	0

Some PLDs have only one type of register, such as the D-type found in most PAL® and GAL® devices. Some PLDs, such as MAPL have all four types available for maximum flexibility. When a design is implemented using PLD software tools, they will often optimize the design based on the available register type. The JK type is the most flexible and can implement all needed functions such as Set, Reset, Hold, and Toggle. The hold function is useful because no further product terms are required to hold the newly transitioned state of a register. When state machine language is used in the software design tools, the JK type register can easily implement the "If-Then-Else" statement. The T-type register implements only the toggle function and requires a reset to bring it to a known state. An active input causes the output to toggle states and it defaults to a hold state. The T-type register is good in counter applications where only a few product terms can implement a very large counter. The RS register is a simpler version of the JK without the toggle function.

There are several timing parameters that affect the integration of the state machine into a system. Minimizing timing constraints of the input signals is an important factor. *Figure 2* is a timing diagram based on the Moore model as illustrated in *Figure 1*.

T_{SU} is the set-up time required by the state registers of the input signals. It must be guaranteed that all inputs and current state feedback information be stable before a T_{SU} time period prior to the clock arriving to avoid any metastability problems. T_h is the time period that the data must be held valid after the clock edge has occurred. These parameters



TL/L/11302-10

FIGURE 2

must have specified maximum values that are as short as possible to guarantee that the inputs get properly latched into the register. T_{CLK} is the clock to output delay of the state registers. The output logic decodes the current state information and forms the outputs to the system. T_{PD} is the propagation delay of this logic to do this decoding function. For controlling the output functions, the T_{CLK} and T_{PD} parameters must be as short as possible as well. A very fast T_{CLK} and T_{PD} ensure that once a state is entered, the function being controlled by the output control logic will be executed as soon as possible.

Determining the overall speed of a state machine is dependent on various timing parameters and implementations. The maximum clock rate is dependent in many cases on the kind of state feedback that is implemented. When buried registers are used in PLDs, the signals are fed back directly to the AND array and do not pass through a highly capacitive I/O pin that would excessively slow the signal down.

When signals are fed back through these pins the maximum clock rate is lowered by about 10%. With I/O feedback, the clock rate is dependent on two timing parameters, T_{SU} and T_{CLK}. These two combine together to form the time period of the state information being clocked through the register (T_{CLK}) followed by the time delay through the logic array T_{SU}). Thus the clock frequency is $F_{max} = 1/(T_{SU} + T_{CLK})$. When there is no feedback at all, the set up times aren't an issue and the flip-flops can run much faster. However, in a state machine this clock rate is misleading because there will be feedback in the circuit. The MAPL128, for example, has a clock rate of 62.5 MHz with no feedback and a 40 MHz speed using I/O feedback. The more typical situation is when the state bits are stored in the buried registers which allows the device to operate at a higher clock rate of 45.5 MHz like in MAPL144. Many programmable logic devices, especially the newer, high density varieties, specify a maximum clock rate that the device will operate at without feedback. In a general logic replacement application, this may be suitable but with the inherent feedback in state machines, it should be questioned if those devices were designed with this application in mind. The MAPL devices are specified with respect to this buried feedback. This parameter, as all others, should be viewed from the system perspective of the actual application. There is an architecture dependence on performance as well. In certain devices, having one too many product terms can halve the maximum clock frequency the device will run at. These architectures will be discussed later.

THE STATE MACHINE DESIGN PROCESS

The state machine design process follows a specific procedure and has been simplified greatly with today's powerful design tools.

1. Description of the sequential circuit
2. State Diagram
3. Solving asynchronous design and metastability problems
4. State/transition tables
5. Minimization
6. Flip-flop determination and equation generation
7. PLD selection and programming

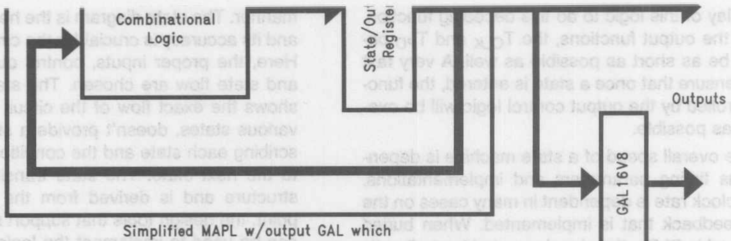
This process defines a procedure for implementing state machine designs. Describing the sequential circuit involves examining the timing diagrams, protocols, etc. that define the exact nature of the problem. The state diagram is the actual solution to the problem described in a flow-chart type manner. The state diagram is the heart of the state machine and its accuracy is crucial for the circuit to operate properly. Here, the proper inputs, control outputs, clock frequency, and state flow are chosen. The state diagram, although it shows the exact flow of the circuit as it steps through the various states, doesn't provide a structured format for describing each state and the conditions required to transition to the next state. The state transition table provides this structure and is derived from the state diagram. At this point, the design tools that support state machine language can be used to implement the logic straight from the state diagram or the state transition table. This makes for an easy transition to the actual implementation in the MAPL device. Traditionally, once the state diagram and state transition table were derived, the designer had to implement this information into simplified logic equations by way of Karnaugh maps, a manual process that was time consuming and only practically suitable for a small number of state bits and input signals. Equations may still be used to derive the output control functions, but today design software is able to take the information directly from the state diagram and state transition table and generate simplified equations in a common file format that different software tools can easily access and manipulate.

Once the state machine has been properly implemented in the PLD device, the output control equations are derived from the state diagram or the state tansition table. There are a few ways of doing this. One way is to use an external PLD with combinational logic or the PAL array that is internal to the MAPL2 family. This combinational logic would then decode the state information for each of the output functions. This is illustrated in the Mealy and Moore models as described above and in the MAPL2 family illustrated in *Figure 3*.

Another way to implement the output functions that are common with PLDs, is to add extra state bits and have the state bits implement the actual output functions, as shown in *Figure 4*.

This implementation assigns state bits that identically represent the needed output coding, hence the state bits become the output signals. By eliminating the output control logic, the T_{PD} parameter is no longer a factor and faster state machine performance will result. The designer has to ensure that there are enough state bits and that they correspond to the needed outputs. If more outputs are needed than state bits available, duplicating next-state logic equations or increasing the number of state bits and assigning them to unused outputs in the PLD will ensure the right number of output control signals.

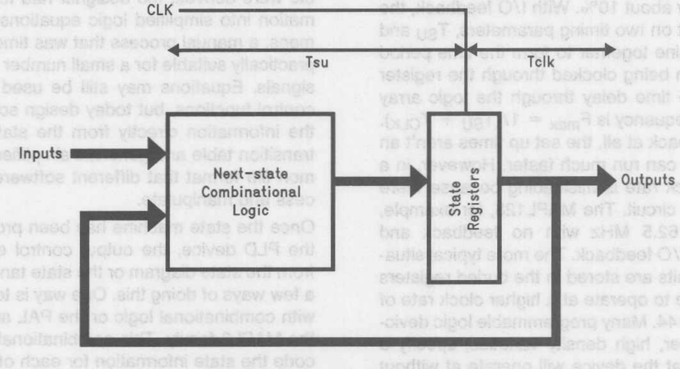
Figure 5 shows an example of a state diagram. A state diagram is usually based on a detailed timing analysis of the circuit being implemented; however, for this simple example the timing analysis will be skipped.



Simplified MAPL w/output GAL which decodes states for output functions.

TL/L/11302–3

FIGURE 3



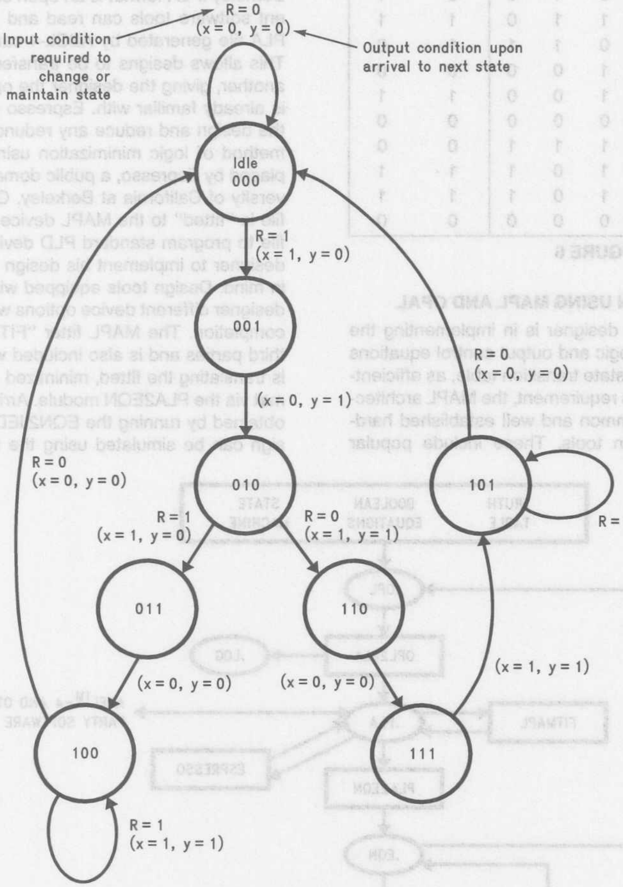
TL/L/11302–4

Output bits the same as state bits if possible

FIGURE 4

The state diagram *(Figure 5)* is the core of the state machine circuit. It defines the inputs, outputs and the conditions required to make a state transition. The diagram assumes a specific clock frequency to ensure proper timing parameters are met. The time between state transitions is the period of this clock. Within each bubble, the state is coded in binary. Each binary bit corresponds to a state register. The name of the state may be included as well if desired for reference purposes. The various paths the state machine may take are indicated by the arrows. The transitions (arrows) are labeled by the input signal conditions needed to cause that transition and by the values the outputs will take after the transition (enclosed in parenthesis).

Once the sequential circuit has been described and completely defined in the state diagram, the state transition table is derived to facilitate the PLD implementation. The table is essentially a modified truth table that extracts from the state diagram all of the state and input conditions required to make a transition and the output signal values based on the newly transitioned state. Every possible combination of signals is defined here. The table is created by starting with a state, usually a default or reset state, and examining what conditions cause a state transition. These conditions could be a change in an input signal value, or an unconditional transfer where there are no branches and the next state is transferred to regardless of input conditions. This process is then done for all the remaining states and the output signal values are defined for each of the new states. The state bits are the actual register outputs in the implemented design.



TL/L/11302-5

FIGURE 5

]

Every state transition and input condition is listed. It should become quite apparent, when creating the state transition table, that its accuracy is entirely dependent upon that of the state diagram. The state transition table for this example is shown in *Figure 6*.

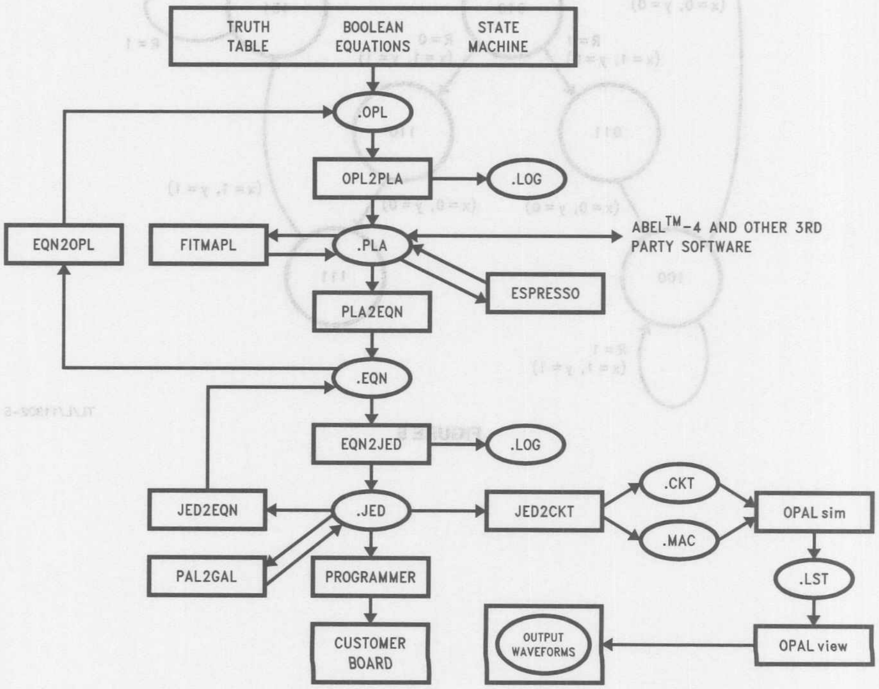
Current State			Inputs	Next State			Outputs	
S2	S1	S0	R	S2	S1	S0	X	Y
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	1	0
0	0	1	X	0	1	0	0	1
0	1	0	0	1	1	0	1	1
0	1	0	1	0	1	1	1	0
0	1	1	X	1	0	0	0	0
1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	0	0	0
1	1	0	X	1	1	1	0	0
1	1	1	X	1	0	1	1	1
1	0	1	1	1	0	1	1	1
1	0	1	0	0	0	0	0	0

FIGURE 6

STATE MACHINE DESIGN USING MAPL AND OPAL

An important need of the designer is in implementing the next-state combinational logic and output control equations from the state diagram, or state transition table, as efficiently as possible. To meet this requirement, the MAPL architecture allows the use of common and well established hardware and software design tools. These include popular third-party tools such as ABEL-4 and CUPL. National Semiconductor's own OPAL-Open Programming Architecture Language is a powerful, open software tool for designing with PAL and GAL devices, as well as the MAPL family. *Figure 7* shows a flow chart of the design process using OPAL.

OPAL allows the designer to describe a design with Boolean equations, truth tables and state machine entry format. OPAL features pull-down menus, a text editor and an automatic fitter that are all easy to use. The OPAL source file is in the .OPL format. The OPL2PLA module compiles this .OPL file to the industry standard Berkeley .PLA format. The Berkeley .PLA format is an open standard format that different software tools can read and process. For example, a PLA file generated by ABEL-4 can also be read by OPAL. This allows designs to be transferred from one tool set to another, giving the designer the option to use tools that he is already familiar with. Espresso can be used to minimize the design and reduce any redundant logic. The traditional method of logic minimization using Karnaugh maps is replaced by Espresso, a public domain program from the University of California at Berkeley. Once minimized, the .PLA file is "fitted" to the MAPL device or compiled to a JEDEC file to program standard PLD devices. The fitter allows the designer to implement his design without a specific device in mind. Design tools equipped with various fitters give the designer different device options when the design is nearing completion. The MAPL fitter "FITMAPL" is available from third parties and is also included with OPAL. The next step is translating the fitted, minimized .PLA file to the .EQN format via the PLA2EQN module. Arriving at the JEDEC map is obtained by running the EQN2JED module. Finally, the design can be simulated using the modules as described in



TL/L/11302–14

FIGURE 7

the OPAL flow chart. With OPAL, the modules can be executed individually, or the sequence can be automated using the translate command. This allows a quick analysis of the design as it is being implemented.

Other important features of OPAL are the PAL2GAL, JED2EQN and EQN2OPL modules. The PAL2GAL module converts the JEDEC fuse map of a wide variety of bipolar PALs to the JEDEC fuse map required by GALs. GALs are growing in popularity due to their electrically erasable CMOS process, low power consumption, and macrocell flexibility that allow one GAL device to replace multiple PAL devices. Refer to the PAL to GAL conversion application note that describes this process in more detail. JED2EQN and EQN2OPL modules convert an arbitrary JEDEC file to the OPAL source format. This allows multiple PAL/GAL designs to be combined in a single MAPL device. Each GAL JEDEC file is converted to the OPAL language. The individual files are then merged in a single text file to be processed by OPAL and fitted to a single MAPL device.

THE MAPL STATE MACHINE ARCHITECTURE ADVANTAGE

The latest generation of PLDs today are able to integrate the logic functions that before would require several of the simpler 20 and 24 pin PLDs. However, these newest devices have significant limitations and may not meet all of the designers needs. Most complex PLDs are designed for general purpose logic replacement and often don't have the features needed for high speed, complex state machine design. FPGAs and many complex PLDs have focused on integration at the expense of overall system speed. Some integrate dozens of product terms throughout the chip, but each output cell may have access to only a few, a severe limitation in state machines where the total number of product terms *and* the number per output are both important. To increase the number of product terms per output some devices fold back buried arrays, adding logic levels and increasing delays. Programmable gate arrays, while offering flexibility also need to be routed and predetermined internal delays may be unknown. The increase in density also brings with it performance limitations due to longer and more capacitive routing channels.

For high performance state machine applications, the new MAPL family of programmable logic is designed to solve these problems. MAPL is designed specifically for implementing state machines. By focusing on a particular application, the device gives the designer important benefits. MAPL achieves these through the use of a proprietary State Machine Architecture that is optimized for state machine implementation.

FPLA ARRAY STRUCTURE

Programmable logic devices can be designed with either the PAL or the FPLA structure. While the PAL architecture is very popular for absorbing random logic, the Field Programmable Logic Array-FPLA is the best architecture where a large number of product terms are needed for each output function.

Both the PAL and the FPLA are composed of an "AND" array and an "OR" array. The "AND" array is essentially a large number of multiple input AND gates whose inputs come from external sources or internal feed back signals. This AND array is programmable. This means that each AND gate can have a programmable number of input variables from the array's pool of input sources. This programmable array gives flexibility to use only the inputs needed to define a particular function. The OR array consists of a large number of multi-input OR gates. The macrocells consist of various types of flip-flops (D/E, J/K, R/S, or T type) and/or combinational logic outputs. The macrocells may be buried; the outputs are fed back to the AND array and don't have a direct output to a pin. They can also be dedicated as an output or used as both an input and output.

The PAL has a fixed OR array whereas the FPLA has a programmable OR array. Each OR gate in a PAL has a fixed number of product term inputs. The FPLA has a programmable number of inputs to the OR gate. Not only does the FPLA have this programmable array, but the product terms defined in the AND array can be shared among the different functions. A product term used in output function F2, for example, can be shared by output F3. By sharing product terms, the FPLA is very efficient in logic utilization where product terms are needed by various macrocells within the device. In a PAL by contrast, output F3 would have to duplicate the product term used by output F2.

Since the FPLA shares product terms across the device, each product term is defined only once and each output has access to it. For example, MAPL has 128 total product terms and each output or buried macrocell has access to all of them. Because of this sharing of product terms, the state machine is best implemented in a FPLA architecture. In a state machine there are multiple state bits which use common product terms based on the current state and/or input conditions to cause transitions. This can be seen in the state transition table as defined in the previous example. In complex applications where dozens of product terms are shared by various state bits, the PAL structure would be prohibitively large if it were to integrate that many product terms because of the large amount of duplication that would occur. While the FPLA is more flexible, the propagation delay through the programmable OR array increases and could cause set-up time constraints with external input signals. The sharing of product terms by the state bits and output functions means product term utilization is most efficient with the FPLA. A PAL array of the same 128 product term per output density as MAPL would be the equivalent of more than 4 GAL22V10s.

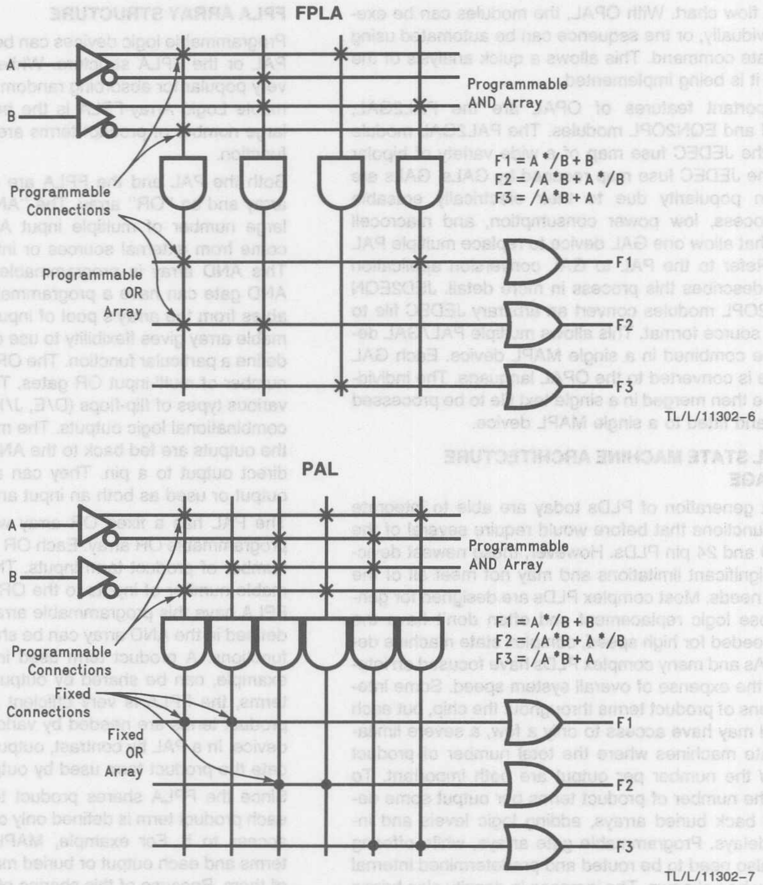


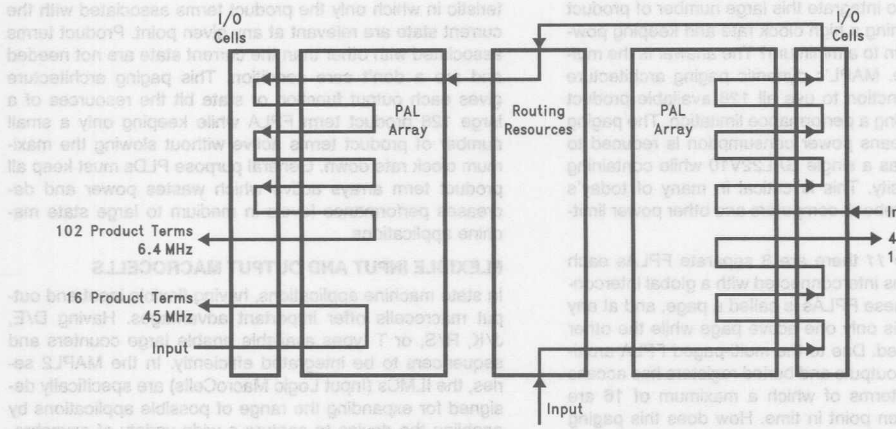
FIGURE 8

MULTIPAGED ARCHITECTURE

Whereas most new architectures make compromises to fit a broad range of applications, the MAPL architecture is optimized for state machine applications. *Figure 9* is a popular architecture that integrates multiple PAL blocks with routing resources.

While *Figure 10* is a generalization, some important characteristics should be noted. The architectures have focused on a PAL architecture with a small, fixed number of product terms per macrocell array. If a macrocell array isn't large enough, to increase the number of product terms for a particular function the signal paths must travel through multiple arrays to gather the required number of product terms. This brings with it performance degradations that are proportional to the number of arrays traversed. A path through two 15 ns arrays would increase the propagation delay to 30 ns or more if delays through the routing resources are included. A function that uses 16 or fewer product terms can operate full speed at 45 MHz, however, a function needing more product terms must be implemented by two or more PAL blocks arranged serially. Thus, if 41 product terms are needed for a function, then three trips through the array are needed, decreasing the maximum clock rate by a factor of 3 to 15 MHz. The MAPL architecture, on the other hand, has the same high clock rate regardless of the number of product terms used for a function, up to the maximum in the device.

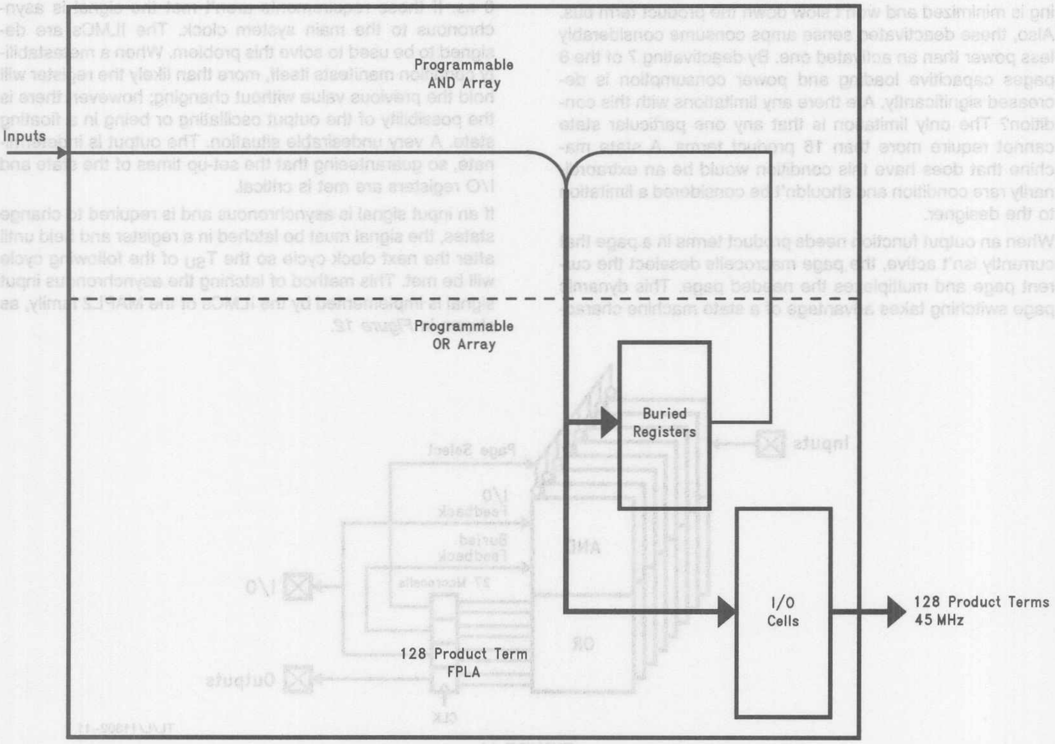
Not only is there a performance limitation with the architecture in *Figure 9*, but fitting the logic equations to the specific device can be excessively complex. When product terms are needed by one function from an adjacent PAL array, assigning signals to pins becomes very difficult and in some applications may be impossible. This may force the designer to use a larger, more expensive device, even though it's full I/O and logic capabilities are underutilized. MAPL, on the other hand is very flexible in its fitting process, allowing the designer to optimize pin assignment that would best suit the application.



Adding product terms for an output function by cascading multiple PAL arrays together. Note dramatic decrease in performance as additional logic levels are added.

TL/L/11302–8

FIGURE 9



TL/L/11302–9

The MAPL128 Architecture. Each macrocell has access to all 128 product terms in th FPLA. Speed is constant irregardless of number of product terms per function.

FIGURE 10

How is MAPL able to integrate this large number of product terms while maintaining a high clock rate and keeping power consumption down to a minimum? The answer is the multipaged architecture. MAPL's dynamic paging architecture allows an output function to use all 128 available product terms without incurring a performance limitation. The paging mechanism also means power consumption is reduced to 110 mA, the same as a single GAL22V10 while containing four times the density. This is critical in many of today's battery operated notebook computers and other power limited designs.

As shown in *Figure 11* there are 8 separate FPLAs each with 16 product terms interconnected with a global interconnect bus. Each of these FPLAs is called a page, and at any point in time, there is only one active page while the other seven are deactivated. Due to the multi-paged FPLA architecture, each of the outputs and buried registers has access to all 128 product terms of which a maximum of 16 are available at any given point in time. How does this paging architecture increase the maximum clock rate and decrease the maximum current consumption? When a page is active, sense amps from the AND array drive the product terms into the OR array. These activated sense amps consume power and contribute a capacitive load to the OR array. When the sense amps are deactivated (tristated) their capacitive loading is minimized and won't slow down the product term bus. Also, these deactivated sense amps consume considerably less power than an activated one. By deactivating 7 of the 8 pages capacitive loading and power consumption is decreased significantly. Are there any limitations with this condition? The only limitation is that any one particular state cannot require more than 16 product terms. A state machine that does have this condition would be an extraordinarily rare condition and shouldn't be considered a limitation to the designer.

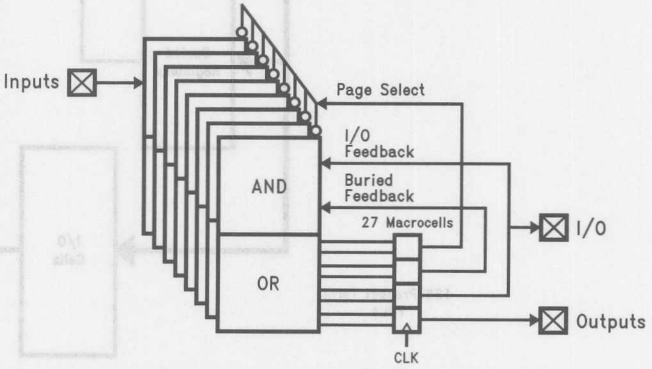
When an output function needs product terms in a page that currently isn't active, the page macrocells deselect the current page and multiplexes the needed page. This dynamic page switching takes advantage of a state machine characteristic in which only the product terms associated with the current state are relevant at any given point. Product terms associated with other than the current state are not needed and are a don't care condition. This paging architecture gives each output function or state bit the resources of a large 128 product term FPLA while keeping only a small number of product terms active without slowing the maximum clock rate down. General purpose PLDs must keep all product term arrays active which wastes power and decreases performance levels in medium to large state machine applications.

FLEXIBLE INPUT AND OUTPUT MACROCELLS

In state machine applications, having flexible input and output macrocells offer important advantages. Having D/E, J/K, R/S, or T types available enable large counters and sequencers to be integrated efficiently. In the MAPL2 series, the ILMCs (Input Logic MacroCells) are specifically designed for expanding the range of possible applications by enabling the device to capture a wide variety of asynchronous input signals.

Input signals and current state information logically combine to determine the next state. As mentioned previously these input signals must meet certain set-up and hold times for the registers. For MAPL, the $T_{SU} = 17$ ns while the $T_{hld} = 0$ ns. If these requirements aren't met the signal is asynchronous to the main system clock. The ILMCs are designed to be used to solve this problem. When a metastability condition manifests itself, more than likely the register will hold the previous value without changing; however, there is the possibility of the output oscillating or being in a floating state. A very undesirable situation. The output is indeterminate, so guaranteeing that the set-up times of the state and I/O registers are met is critical.

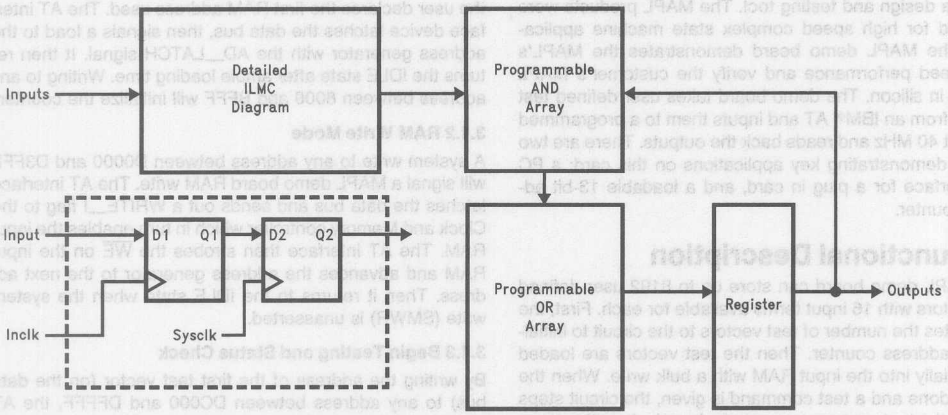
If an input signal is asynchronous and is required to change states, the signal must be latched in a register and held until after the next clock cycle so the T_{SU} of the following cycle will be met. This method of latching the asynchronous input signal is implemented by the ILMCs of the MAPL2 family, as shown in *Figure 12*.



TL/L/11302-11

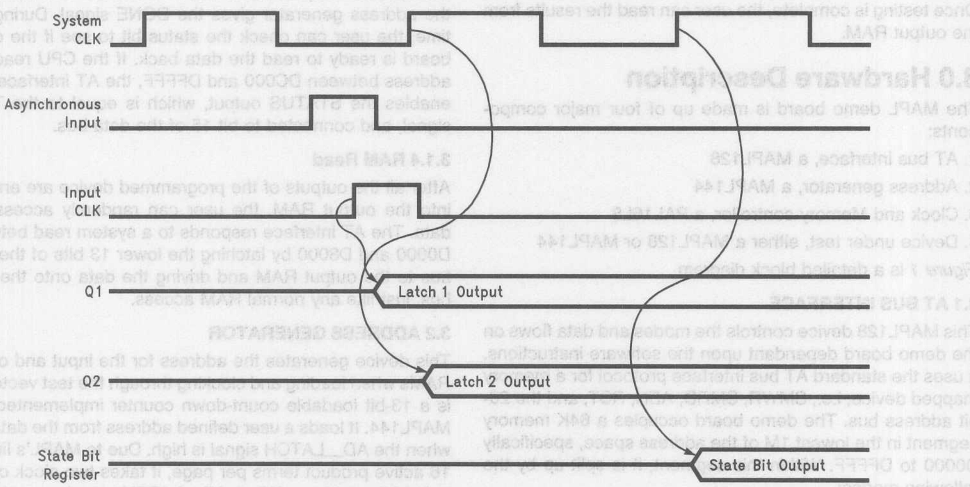
FIGURE 11

The ILMCs provide double buffering of 16 inputs coming from the input pins. Either single, double, or no buffering at all can be selected via the select mux. The ILMC registers may be preset via an intialization product term and the input data can be clocked by three selectable clock sources: the system clock, a separate input clock, and by a global product term shared by all ILMCs. The solution for metastability prevention is to synchronize the asynchronous signal by preventing it from appearing until after the system clock so it will have the proper T_{SU} for the following system clock. This is done by first latching the signal in a register, probably using the input clock or global product term, and then isolating the latched output using a second buffer which is clocked by the system clock. The first register must be isolated from the AND array because if it's output occurs too soon, the T_{SU} of the next clock cycle may still not be met. By clocking the second register by the system clock, it will be guaranteed that the T_{SU} of the following clock cycle will be met.



TL/L/11302–12

FIGURE 12



TL/L/11302–13

FIGURE 13

MAPL™ Demonstration Board

National Semiconductor
Application Note 802
David O'Neal

1.0 Introduction

The goals of this project are to show the versatility and functionality of National Semiconductor's Multipaged Array of Programmable Logic, MAPL, as well as provide the customer with a design and testing tool. The MAPL products were designed for high speed complex state machine applications. The MAPL demo board demonstrates the MAPL's high speed performance and verify the customer's MAPL designs in silicon. The demo board takes user defined test vectors from an IBM® AT and inputs them to a programmed MAPL at 40 MHz and reads back the outputs. There are two MAPLs demonstrating key applications on this card: a PC bus interface for a plug in card, and a loadable 13-bit address counter.

2.0 Functional Description

The MAPL demo board can store up to 8192 user defined test vectors with 16 input terms available for each. First, the user writes the number of test vectors to the circuit to initialize the address counter. Then the test vectors are loaded sequentially into the input RAM with a bulk write. When the write is done and a test command is given, the circuit steps through the test vectors, one per clock cycle, inputting them to the customer's programmed device at 40 MHz. Up to 16 outputs of the tested device are stored in the output RAM with each clock cycle. During the testing process, the CPU can check if the board is still testing with the status bit. Once testing is complete, the user can read the results from the output RAM.

3.0 Hardware Description

The MAPL demo board is made up of four major components:

1. AT bus interface, a MAPL128
2. Address generator, a MAPL144
3. Clock and Memory controller, a PAL16L8
4. Device under test, either a MAPL128 or MAPL144

Figure 1 is a detailed block diagram.

3.1 AT BUS INTERFACE

This MAPL128 device controls the modes and data flows on the demo board dependant upon the software instructions. It uses the standard AT bus interface protocol for a memory mapped device, i.e., SMWR, SMRD, AEN, RST, and the 20-bit address bus. The demo board occupies a 64K memory segment in the lowest 1M of the address space, specifically D0000 to DFFFF. Within this segment, it is split up by the following modes:

D0000—D3FFF	Input RAM on write, output RAM on read
D8000—DBFFF	Address generator initialization
DC000—DFFFF	Begin test on write, circuit status check on read

The OPAL™ listing for the AT interface is listed as *Figure 2*. Following is a description of the AT bus interface modes.

3.1.1 Address Generator Initialization

This is the first mode necessary to set up the circuit. Here the user declares the first RAM address used. The AT interface device latches the data bus, then signals a load to the address generator with the AD__LATCH signal. It then returns the IDLE state after ample loading time. Writing to any address between 8000 and BFFF will initialize the counter.

3.1.2 RAM Write Mode

A system write to any address between D0000 and D3FFF will signal a MAPL demo board RAM write. The AT interface latches the data bus and sends out a WRITE__I flag to the Clock and Memory controller which in turn enables the input RAM. The AT interface then strobes the $\overline{\text{WE}}$ on the input RAM and advances the address generator to the next address. Then it returns to the IDLE state when the system write (SMWR) is unasserted.

3.1.3 Begin Testing and Status Check

By writing the address of the first test vector (on the data bus) to any address between DC000 and DFFFF, the AT interface begins 40 MHz testing. It latches, the data bus and gives a load signal to the address generator (AD__LATCH). It also outputs CLKSEL, signaling the Clock and Memory controller to start the 40 MHz clock, and TEST, to configure the input and output RAMs. The AT interface then waits until the address generator gives the DONE signal. During this time, the user can check the status bit to see if the demo board is ready to read the data back. If the CPU reads an address between DC000 and DFFFF, the AT interface chip enables the STATUS output, which is equal to the TEST signal, and connected to bit 15 of the data bus.

3.1.4 RAM Read

After all the outputs of the programmed device are entered into the output RAM, the user can randomly access this data. The AT interface responds to a system read between D0000 and D3000 by latching the lower 13 bits of the data bus to the output RAM and driving the data onto the data bus, just like any normal RAM access.

3.2 ADDRESS GENERATOR

This device generates the address for the input and output RAMs when loading and clocking through the test vectors. It is a 13-bit loadable count-down counter implemented in a MAPL144. It loads a user defined address from the data bus when the AD__LATCH signal is high. Due to MAPL's limit of 16 active product terms per page, it takes two clock cycles to load the address. When AD__LATCH goes low again, the device counts down to zero, one address per clock cycle. Upon reaching zero, it outputs a DONE flag to signal the AT interface and remains idle. There are 13 inputs for a total of 8K addresses. The OPAL listing is included as *Figure 3*.

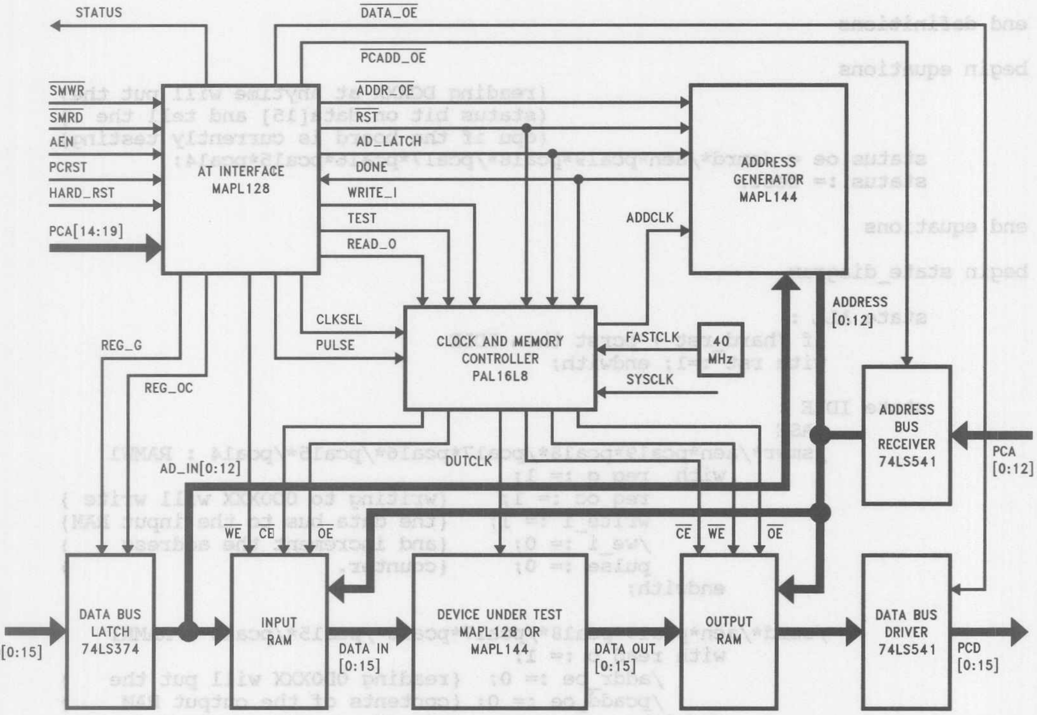
3.3 CLOCK AND MEMORY CONTROLLER

This PAL16L8 device controls the clock timing and enables the memory chips with respect to the AT interface modes. It has two clock inputs, a system clock and one 40 MHz clock, and two clock enable lines, PULSE for the system clock and CLKSEL for the 40 MHz. The system clock is used during the address generator initialization to clock that device through the loading states and prepare it to count. The system clock is also used to decrement the address generator after each RAM write to prepare for the next write. The system clock is inverted to allow the AT interface outputs to settle before clocking. The 40 MHz clock is used in three different phases during the test mode. The first rising edge goes to the address generator to decrement the address. After 8 ns, the address is stable at the input RAM, and the data is stable at the tested device 12 ns later. The MAPL has a 15 ns setup time, thus the device under test needs to be clocked a total of 35 ns (8 + 12 + 15) after the address generator. By using a 16L8B with a nominal propagation time of 11 ns, we can feedback the signal once and generate a clock skewed by 11 ns per cycle, thus giving a rising edge to the customer's design 36 ns (11 + 25 ns clock period) after the address generator. 8 ns after the design is clocked, the data is ready to be written into the output RAM. With a 6.5 ns data setup time. The RAM's write enable can be strobed anywhere from 14.5 ns to 22 ns (address becomes invalid) after the design is clocked. Thus, feeding the design clock back twice, we have the 22 ns delay to strobe the write enable.

The Clock and Memory controller also signals the chip enable and output enable for both input and output RAM. The AT interface device puts out three mode signals, WRITE_I, READ_O, and TEST. On RAM write, the Clock and Memory controller enables the input RAM with disabled outputs; on RAM read, it enables the output RAM with the chip enable, output enable, and not write enabled. In the test mode, the input RAM is read enabled and the output RAM is write enabled. *Figure 4* shows the OPAL listing.

4.0 Summary

The result of this is not only a demonstration of how to use the MAPL products in a typical application, but the demo board also provides customers with a product to test their MAPL designs in silicon. All the device designs were created and simulated using National's OPAL, Open Programming Architecture Language, software. These files and timing diagrams are included.



TL/L/11303-1

FIGURE 1. Block Diagram

```
begin header

        This MAPL128 design is the PC interface for parallel add-in card
        based MAPL demo board.  It is a generic bus manager for a memory
        mapped device.

end header

begin definitions

        device mapl128;

        inputs
                smwr,smrd,aen,pcrst,hard_rst,
                pca19,pca18,pca17,pca16,pca15,pca14,
                done;

        feedbacks (DE,RST)
                test;

        outputs (DE,RST)
                reg_g,/reg_oc,ad_latch,/rst,/addr_oe,
                pulse,status,clksel,write_i,read_o,
                /we_i,/pcadd_oe;

end definitions

begin equations
                                     {reading DCXXX at anytime will put the}
                                     {status bit on data[15] and tell the  }
                                     {cpu if the board is currently testing}
        status.oe = /smrd*/aen*pca19*pca18*/pca17*pca16*pca15*pca14;
        status := test;

end equations

begin state_diagram

        state ALL :
                if /hard_rst + pcrst then IDLE
                with rst :=1; endwith;

        state IDLE :
                CASE
                /smwr*/aen*pca19*pca18*/pca17*pca16*/pca15*/pca14 : RAMW1
                        with  reg_g := 1;
                              reg_oc := 1;      {writing to 0D0XXX will write }
                              write_i := 1;     {the data bus to the input RAM}
                              /we_i := 0;       {and increment the address    }
                              pulse := 0;       {counter.                     }
                        endwith;

                /smrd*/aen*pca19*pca18*/pca17*pca16*/pca15*/pca14 : RAMR1
                        with read_o := 1;
                             /addr_oe := 0;    {reading 0D0XXX will put the   }
                             /pcadd_oe := 0;   {contents of the output RAM    }
                        endwith;               {at address XXX on the data bus}

                /smwr*/aen*pca19*pca18*/pca17*pca16*pca15*/pca14 : ADDW1
                        with reg_g := 1;
```

TL/L/11303–2

FIGURE 2

```
                     reg_oc := 1;    {writing to 0D8XXX will write}
                     pulse :=1;      {the data bus to the address }
                     ad_latch := 1;  {generator.                  }
                endwith;
        /smwr*/aen*pca19*pca18*/pca17*pca16*pca15*pca14 : TEST1
                with reg_g := 1;
                     reg_oc := 1;    {writing to 0DCXXX will write }
                     ad_latch := 1;  {the data bus to the address }
                     test := 0;      {generator and begin test mode}
                endwith;
        ENDCASE;

state RAMW1 :
        if smwr then IDLE
        with /we_i := 1;             {RAM write:                  }
             pulse := 1;             {1 - latch the data bus      }
             write_i := 1;           {2 - write enable the memory }
             reg_oc := 1; endwith    {3 - wait for smwr to go low }
        else RAMW1                   {    (to avoid double write) }
        with /we_i := 1;             {4 - increment addr generator}
             pulse := 0;
             write_i := 1;
             reg_oc := 1; endwith;

state RAMR1 :
        goto RAMR2                   {RAM read:                    }
        with read_o := 1;            {1 - disable addr.gen. address}
             /addr_oe := 0;          {2 - enable PC address        }
             /pcadd_oe := 0;         {3 - read enable output RAM   }
        endwith;                     {    and data bus driver      }
                                     {4 - hold for 2 clocks        }
state RAMR2 :
        goto IDLE with
            /addr_oe := 0; endwith;

state ADDW1 :                        {Address write:              }
        goto ADDW2                   {1 - latch data bus          }
        with reg_oc := 1;            {2 - send ad_latch to addrgen}
             ad_latch := 1;          {    to read address         }
             pulse := 1;             {3 - strobe addrgen twice    }
        endwith;                     {    (two clocks to read)    }
                                     {4 - wait for smwr to go low }
state ADDW2 :
        goto ADDW3
        with reg_oc := 1;
             ad_latch := 1;
             pulse := 1;
        endwith;

state ADDW3 :
        if smwr then IDLE
        with reg_oc := 1;
             ad_latch := 0;
             pulse := 1; endwith
        else ADDW3
        with reg_oc := 1;
             ad_latch := 0;
             pulse := 0; endwith;
```

TL/L/11303-3

FIGURE 2 (Continued)

```
        state TEST1 :
              goto TEST2                          {Test mode:}
              with reg_oc := 1;                   {1 - load address as above }
                   clksel := 1;                   {2 - set addclk to fastclk }
                   ad_latch := 1;                 {3 - assert test flag      }
                   test := 1;                     {4 - wait for done signal  }
              endwith;
        state TEST2 :
              goto TEST3
              with ad_latch := 1;
                   clksel := 1;
                   test := 1;
              endwith;

        state TEST3 :
              if done then IDLE
              else TEST3
                    with clksel := 1;
                         test := 1;
                    endwith;

end state_diagram
```

TL/L/11303–4

FIGURE 2 (Continued)

```
begin header
    address generator for MAPL Demo Board
    This design is a loadable 13-bit countdown counter
    implemented on the MAPL144 device
end header

begin definitions

    device MAPL144;

    inputs
        rst,ad_latch,addr_oe,
        in12,in11,in10,in9,in8,in7,in6,in5,in4,in3,in2,in1,in0;

    statebits (D,RST,BURIED)
        sb2,sb1,sb0;

    feedbacks (JK,HOLD)
        ad12,ad11,ad10,ad9,ad8,ad7,ad6,ad5,ad4,ad3,ad2,ad1,ad0;

    output (D,SET)
        done;

    set count = [ad12,ad11,ad10,ad9,ad8,ad7,ad6,ad5,ad4,ad3,ad2,ad1,ad0];
    set ins = [in12,in11,in10,in9,in8,in7,in6,in5,in4,in3,in2,in1,in0];

end definitions

begin equation

    count.oe = addr_oe;
    done.oe = 1;

end equation

begin truth_table

    ttin
    rst,ad_latch,
    sb2,sb1,sb0,
    ad12,ad11,ad10,ad9,ad8,ad7,ad6,ad5,ad4,ad3,ad2,ad1,ad0,
    in12,in11,in10,in9,in8,in7,in6,in5,in4,in3,in2,in1,in0;

    ttout
    ad12,ad11,ad10,ad9,ad8,ad7,ad6,ad5,ad4,ad3,ad2,ad1,ad0,
    sb2,sb1,sb0,
    done;

    0- ---  ------------- -------------   0000000000000 000 1 {IDLE}
    10 000  ------------- -------------   0000000000000 000 1
    11 000  ------------- -------------   ????????????? 001 1

    1- 001  ------------- 0------------   0------------ 010 1 {LOAD_HI}
    1- 001  ------------- 1------------   1------------ 010 1
    1- 001  ------------- -0-----------   -0----------- 010 1
    1- 001  ------------- -1-----------   -1----------- 010 1
    1- 001  ------------- --0----------   --0---------- 010 1
    1- 001  ------------- --1----------   --1---------- 010 1
    1- 001  ------------- ---0---------   ---0--------- 010 1
    1- 001  ------------- ---1---------   ---1--------- 010 1
```

TL/L/11303-5

FIGURE 3

```
        1- 001  ------------- ----0--------   ----0-------- 010 1
        1- 001  ------------- ----1--------   ----1-------- 010 1
        1- 001  ------------- -----0-------   -----0------- 010 1
        1- 001  ------------- -----1-------   -----1------- 010 1
        1- 001  ------------- ------0------   ------0------ 010 1
        1- 001  ------------- ------1------   ------1------ 010 1

        1- 010  ------------- -------0-----   -------0----- 011 1 {LOAD_LO}
        1- 010  ------------- -------1-----   -------1----- 011 1
        1- 010  ------------- --------0----   --------0---- 011 1
        1- 010  ------------- --------1----   --------1---- 011 1
        1- 010  ------------- ---------0---   ---------0--- 011 1
        1- 010  ------------- ---------1---   ---------1--- 011 1
        1- 010  ------------- ----------0--   ----------0-- 011 1
        1- 010  ------------- ----------1--   ----------1-- 011 1
        1- 010  ------------- -----------0-   -----------0- 011 1
        1- 010  ------------- -----------1-   -----------1- 011 1
        1- 010  ------------- ------------0   ------------0 011 1
        1- 010  ------------- ------------1   ------------1 011 1

        10 011  ------------- -------------   ????????????? 100 0 {WAIT}
        11 011  ------------- -------------   ????????????? 011 0

        11 100  ------------- -------------   ????????????? 001 1 {COUNT}
        10 100  ------------- -------------   ------------! 100 0
        10 100  ------------0 -------------   -----------!- 100 0
        10 100  -----------00 -------------   ----------!-- 100 0
        10 100  ----------000 -------------   ---------!--- 100 0
        10 100  ---------0000 -------------   --------!---- 100 0
        10 100  --------00000 -------------   -------!----- 100 0
        10 100  -------000000 -------------   ------!------ 100 0
        10 100  ------0000000 -------------   -----!------- 100 0
        10 100  -----00000000 -------------   ----!-------- 100 0
        10 100  ----000000000 -------------   ---!--------- 100 0
        10 100  ---0000000000 -------------   --!---------- 100 0
        10 100  --00000000000 -------------   -!----------- 100 0
        10 100  -000000000000 -------------   !------------ 100 0
        10 100  0000000000001 -------------   0000000000000 111 0

        -0 111  0000000000000 -------------   0000000000000 000 1 {END_COUNT}
        11 111  ------------- -------------   ????????????? 001 1

end truth_table
```

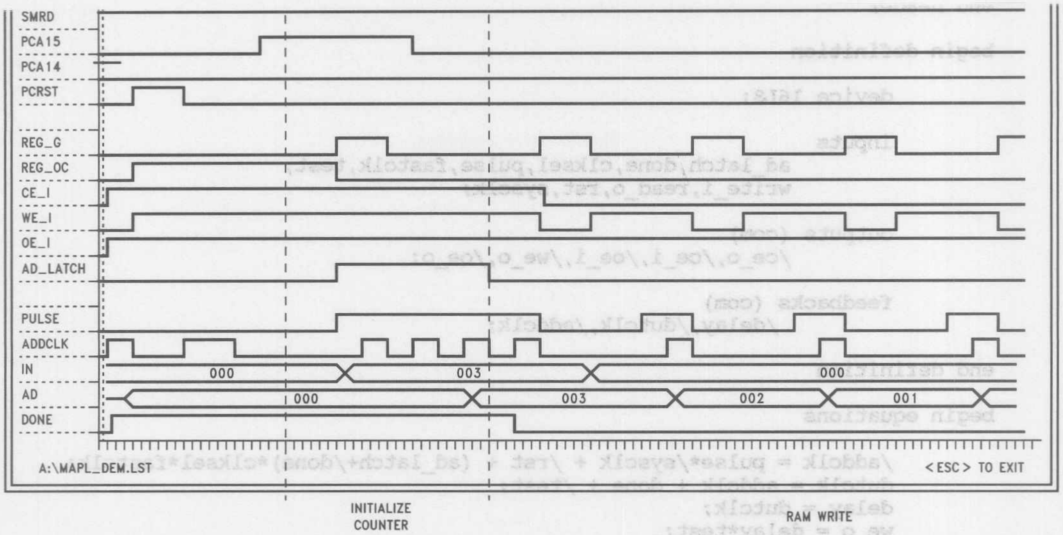
TL/L/11303-6

FIGURE 3 (Continued)

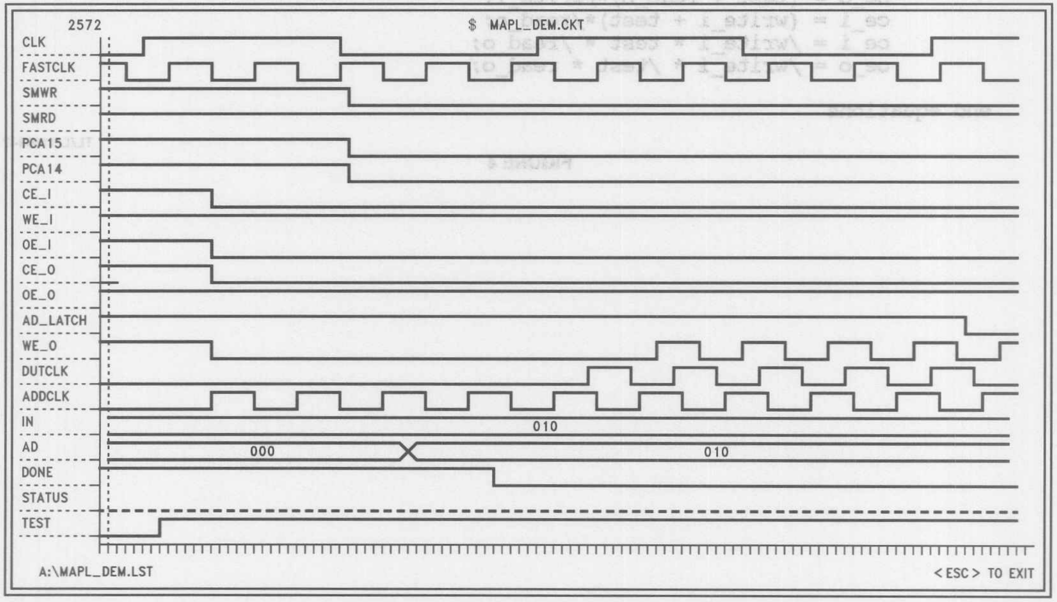
```
begin header
        Clock and Memory controller for MAPL demo board
end header

begin definition

        device 16L8;

        inputs
                ad_latch,done,clksel,pulse,fastclk,test,
                write_i,read_o,rst,sysclk;

        outputs (com)
                /ce_o,/ce_i,/oe_i,/we_o,/oe_o;

        feedbacks (com)
                 /delay,/dutclk,/addclk;

end definition

begin equations

        /addclk = pulse*/sysclk + /rst + (ad_latch+/done)*clksel*fastclk;
        dutclk = addclk + done + /test;
        delay = dutclk;
        we_o = delay*test;

        ce_o = (test + read_o)*/write_i;
        ce_i = (write_i + test)*/read_o;
        oe_i = /write_i * test * /read_o;
        oe_o = /write_i * /test * read_o;

end equations
```

TL/L/11303-7

FIGURE 4

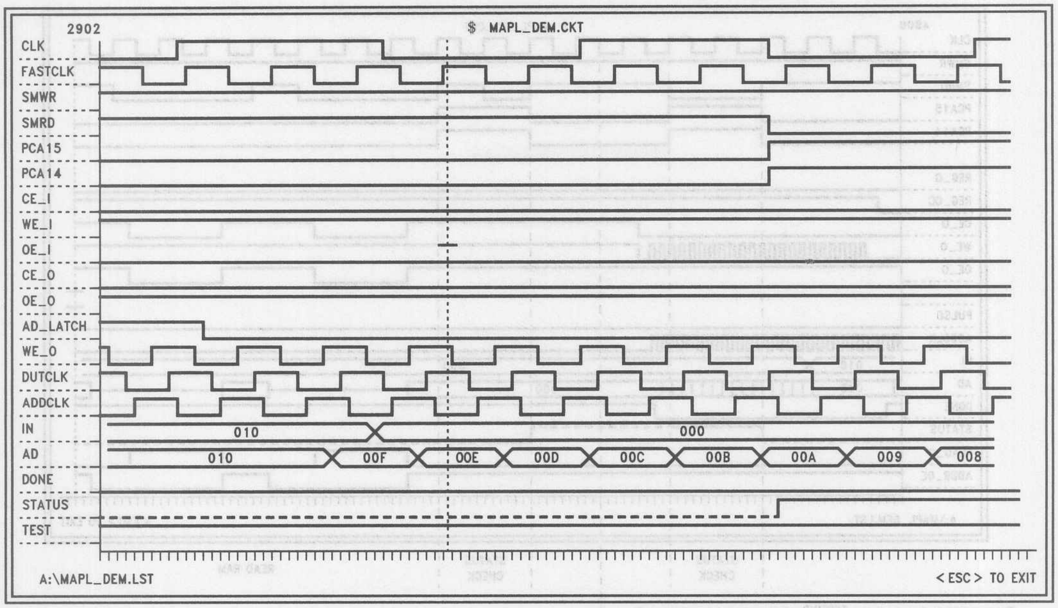


TL/L/11303-8



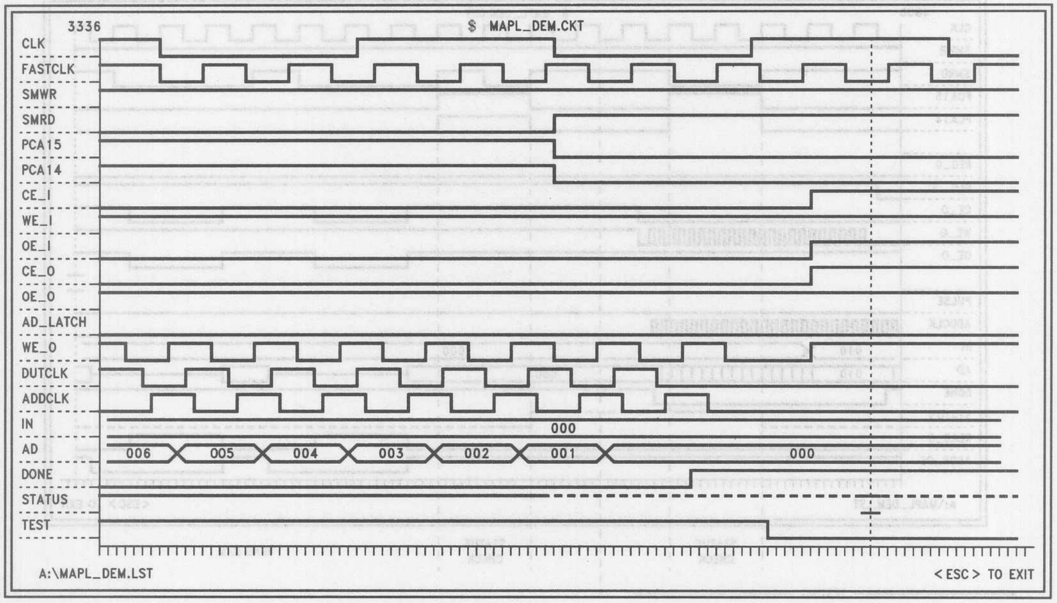
TL/L/11303-9

Beginning of Test



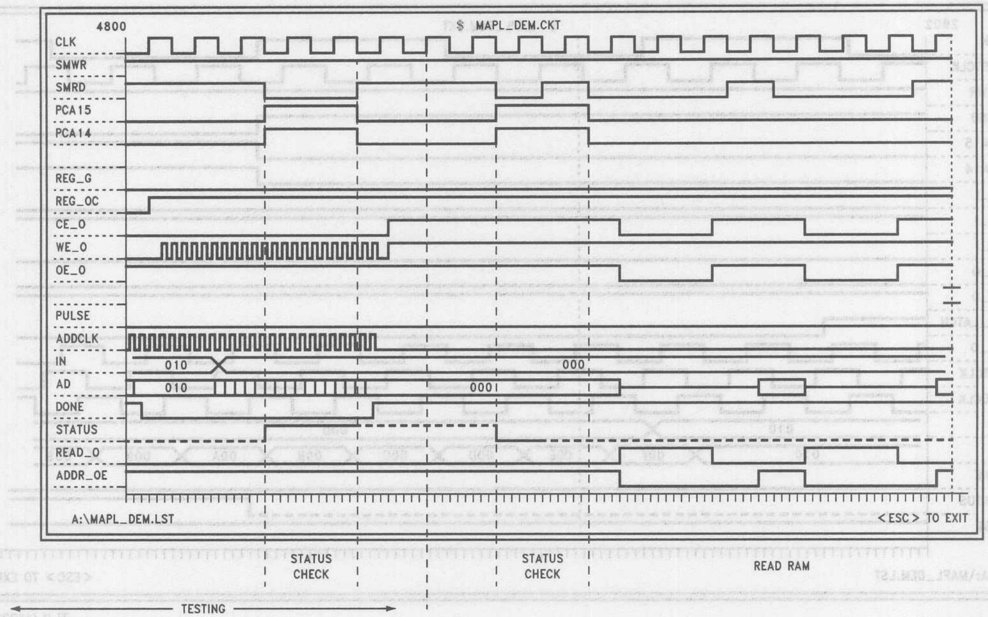
TL/L/11303-10

Testing

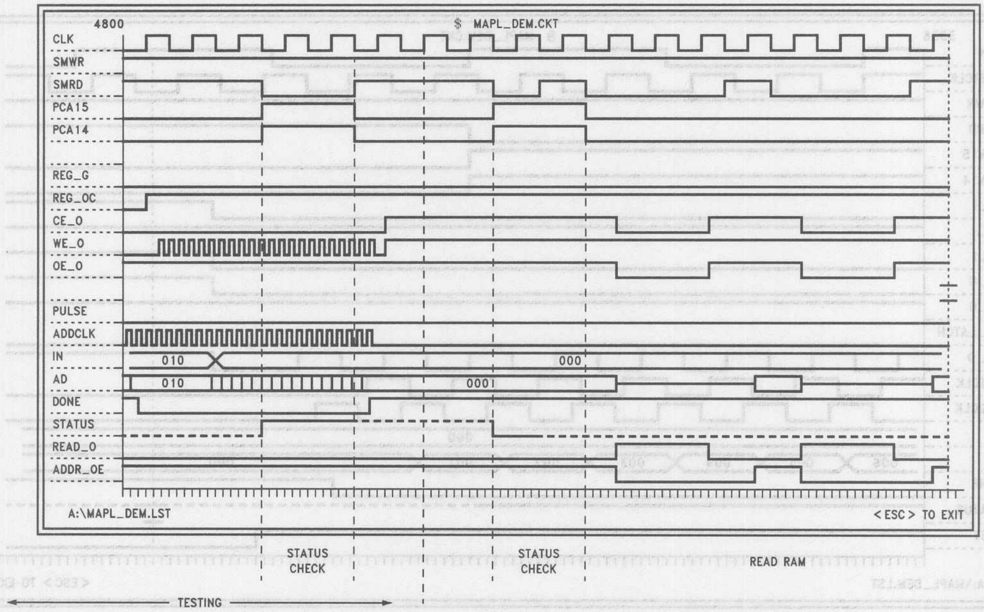


TL/L/11303-11

End of Test

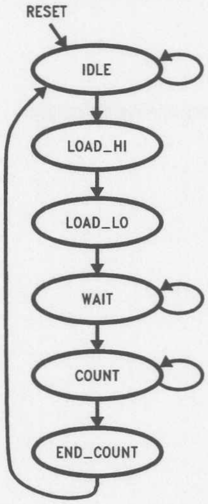


TL/L/11303-12



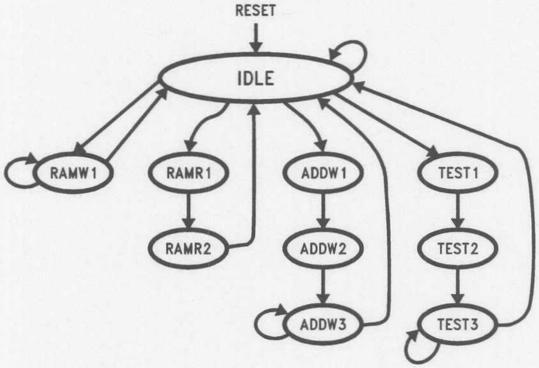
TL/L/11303-13

Address Generator



TL/L/11303-14

PC Interface



TL/L/11303-15

Section 4
PLD Development Tools

Section 4 Contents

OPAL PLD Development Software 4-3
3rd Party Software 4-15
PLD Programming and Testing 4-17

National Semiconductor

OPAL™ Open Programmable Architecture Language PLD Development Software

General Description

The OPAL software package is a comprehensive PLD (Programmable Logic Device) development design tool offered by National Semiconductor. It supports state machine, truth table and Boolean equation entry, as well as optimization, verification and implementation in a wide variety of PLDs.

The OPAL software package consists of: a graphical shell environment, executable modules, a graphical simulation package, a device library file, examples and an overall demonstration of the software package.

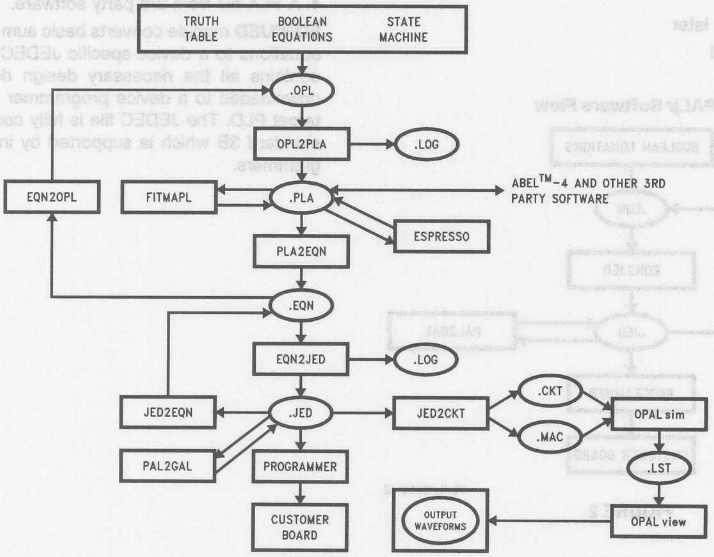
The example entry files contain circuit designs which are written in state machine, truth table and equation file formats. The entry files are used by the modules to create JEDEC maps, which contain the programming data for a target device. The designer can implement new designs by creating new entry files, or by borrowing from existing designs in a variety of file formats.

Features

- Supports all NSC MAPL™/GAL®/ECL PAL® devices
- User friendly menu shell
- Multiple entry formats
- Device independent entry
- Minimization
- Open design environment
- Logic simulation viewer
- Extensive error checking
- Automatic pinlist generation
- Full design documentation
- PAL to GAL conversion
- Demonstration of software package
- Windows 3.0, Sun4 Support

OPAL Software Diagram

OPAL Software Flow



TL/L/9989-1

FIGURE 1

OPAL Software Chart

Software Modules	OPAL PC	OPAL UNIX®	OPALjr PC
OPAL2PLA	X	X	
FITMAPL	X	X	
PLA2EQN	X	X	
EQN2JED	X	X	X
JED2EQN	X	X	X
EQN2OPL	X	X	
PAL2GAL	X	X	X
ESPRESSO	X	X	
JED2CKT	X		
OPALsim	X	X	
OPALview	X		
SHELL	X		X

FIGURE 3

OPALjr

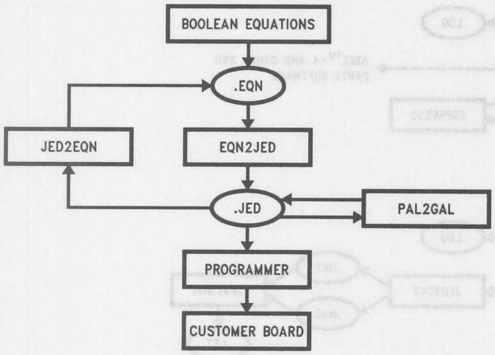
OPALjr is a subset of and is fully compatible with the complete high-level OPAL PLD development software. OPALjr contains five comprehensive OPAL demonstrations and is available for use as equation entry PAL/GAL design software and OPAL demo. OPALjr allows the designer to go from Boolean equation design description to a JEDEC file and also includes the PAL2GAL module to upgrade JEDEC files from PAL to GAL format.

Hardware Requirements

To run OPAL or OPALjr software you will need:

- IBM® PC-AT®, PC-XT® or compatible, with at least 350k RAM
- VGA, EGA, Hercules®, (CGA can be used without waveform viewer)
- MS-DOS® 2.1 or later
- Mouse supported

OPALjr Software Flow



TL/L/9989-2

FIGURE 2

Open Architecture

OPAL is an open architecture language. This means the software is modularized so that there is a standard interface format between modules. These standard interface formats allow the modules to communicate with 3rd party software, thereby letting the user use existing, familiar software tools in conjunction with OPAL. While OPAL is a powerful standalone PLD development tool, it complements the user's existing tools (rather than superceding them) to form an even more powerful and flexible toolbox. The following three sections address these modules in more detail.

Functional Description

User Interface: OPAL includes a user friendly menu driven shell with built-in help and debugging support. The shell allows the user to compile individual modules *(Figure 4)*, or lets OPAL compile the design in one step *(Figure 5)*. During compilation, many of the modules document compilation information and errors to a .LOG file. This is done so the designer can inspect and record the compilation process. Files can be edited with an on-board text editor or the external editor of choice.

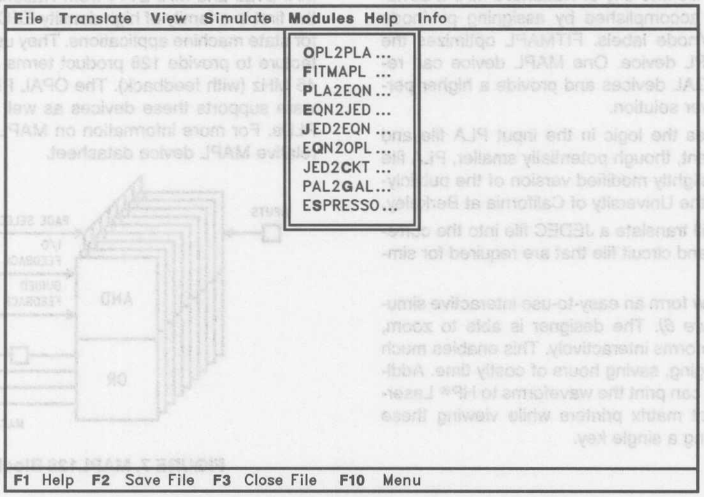
OPL2PLA compiles an OPAL source file (.OPL) and produces an OPEN-PLA file as output. The OPAL source file can contain a mixture of state machine, truth table and Boolean equation definition blocks. The OPEN-PLA format (.PLA) is an industry standard allowing interface to 3rd party software tools.

PLA2EQN module translates an OPEN-PLA format file to a standard sum-of-products (SOP) Boolean equation file. The input PLA file can be:

1. The output of the OPL2PLA module.
2. The output of the ESPRESSO minimizer.
3. The output of the FITMAPL fitting module.
4. A PLA file from 3rd party software.

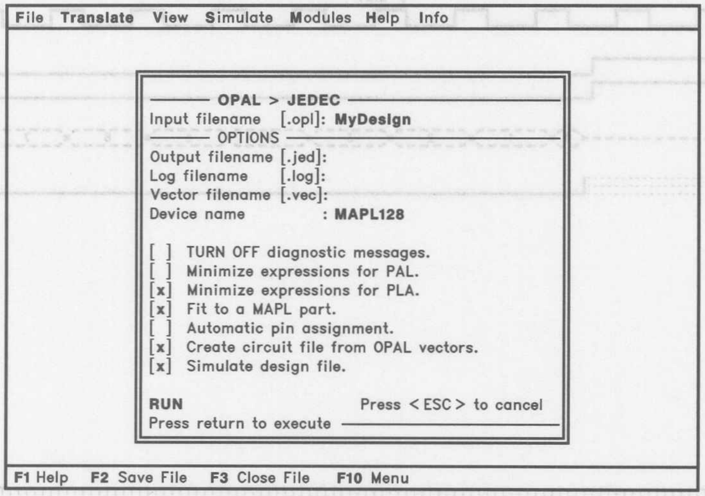
EQN2JED module converts basic sum-of-products Boolean equations to a device-specific JEDEC file. The JEDEC file contains all the necessary design details which can be downloaded to a device programmer for programming the target PLD. The JEDEC file is fully compatible with JEDEC standard 3B which is supported by industry standard programmers.

Functional Description (Continued)



TL/L/9989–3

FIGURE 4. OPAL Menu: OPAL Modules


File Translate View Simulate Modules Help Info
F1 Help F2 Save File F3 Close File F10 Menu

```
——— OPAL > JEDEC ———
Input filename   [.opl]: MyDesign
——————— OPTIONS ———————
Output filename [.jed]:
Log filename    [.log]:
Vector filename [.vec]:
Device name           : MAPL128

[ ]  TURN OFF diagnostic messages.
[ ]  Minimize expressions for PAL.
[x]  Minimize expressions for PLA.
[x]  Fit to a MAPL part.
[ ]  Automatic pin assignment.
[x]  Create circuit file from OPAL vectors.
[x]  Simulate design file.

RUN                        Press <ESC> to cancel
Press return to execute ——————————————
```

TL/L/9989–4

FIGURE 5. OPAL Menu: OPAL Entry File to Simulation

JED2EQN module will disassemble a JEDEC file into the corresponding basic Boolean equations. This is useful when taking existing designs in JEDEC file format and modifying them and/or combining them with other designs. JED2EQN assigns meaningful labels to all the pins of the device. The labels used in the basic Boolean equations created by JED2EQN contain the pin number preceded by the type of signal. Observing labels makes it easy to determine if the pin is used as a dedicated input, combinatorial or registered output, and whether or not the output is used as feedback into the device.

EQN2OPL module converts basic Boolean equations specified in EQN format to an OPAL (.OPL) format file. EQN2OPL is useful when modifying or combining existing design, since it allows the designer modify the design in the more powerful OPAL format as opposed to the simpler equation format.

PAL2GAL module converts a PAL JEDEC file into a GAL JEDEC file. PAL2GAL first checks to ensure that the PAL is replaceable by a supported GAL before it proceeds to do the conversion. PAL2GAL allows the user convert from fused, bipolar PAL devices to the more flexible and lower power CMOS GAL devices.

Functional Description (Continued)

FITMAPL fits a PLA file into any of National's MAPL devices. The process is accomplished by assigning pin/node numbers to the pin/node labels. FITMAPL optimizes the PLA file for the MAPL device. One MAPL device can replace multiple PAL/GAL devices and provide a higher performance, lower power solution.

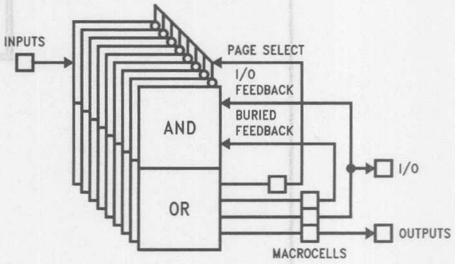
ESPRESSO minimizes the logic in the input PLA file and produces an equivalent, though potentially smaller, PLA file as output. This is a slightly modified version of the publicly-available utility from the University of California at Berkeley.

JED2CKT module will translate a JEDEC file into the corresponding macro file and circuit file that are required for simulation.

OPALsim/OPALview form an easy-to-use interactive simulation package *(Figure 6)*. The designer is able to zoom, pan, and group waveforms interactively. This enables much more efficient debugging, saving hours of costly time. Additionally, the designer can print the waveforms to HP® Laserjet® and Epson® dot matrix printers while viewing these waveforms by pressing a single key.

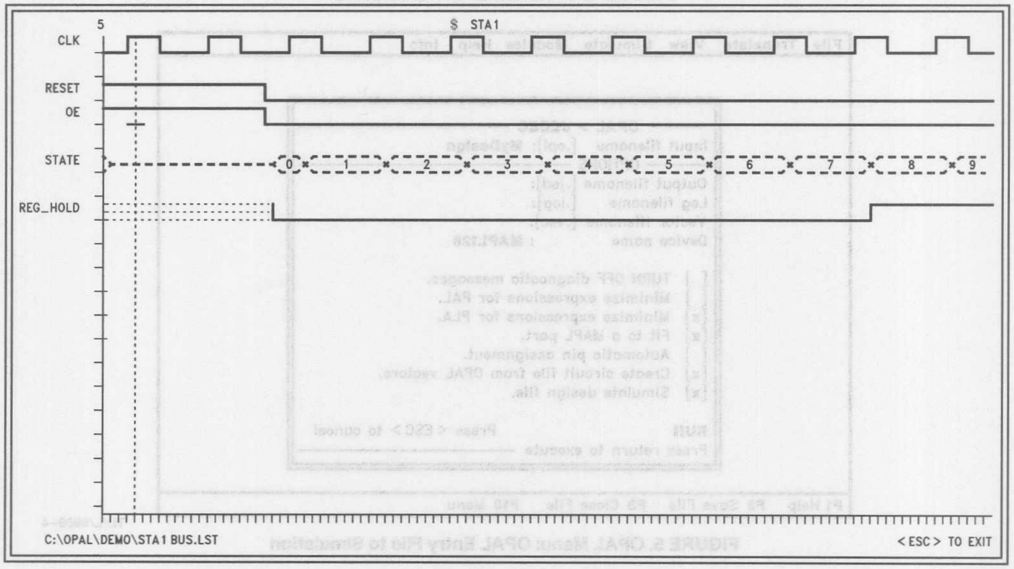
A Word About MAPL

MAPL128 and MAPL144 from National Semiconductor are the first in a family of high density EECMOS PLDs optimized for state machine applications. They use a paged PLA architecture to provide 128 product terms at a system speed of 45 MHz (with feedback). The OPAL PLD development software supports these devices as well as all other National PLDs. For more information on MAPL devices, consult the relative MAPL device datasheet.



TL/L/9989-6

FIGURE 7. MAPL128 Block Diagram

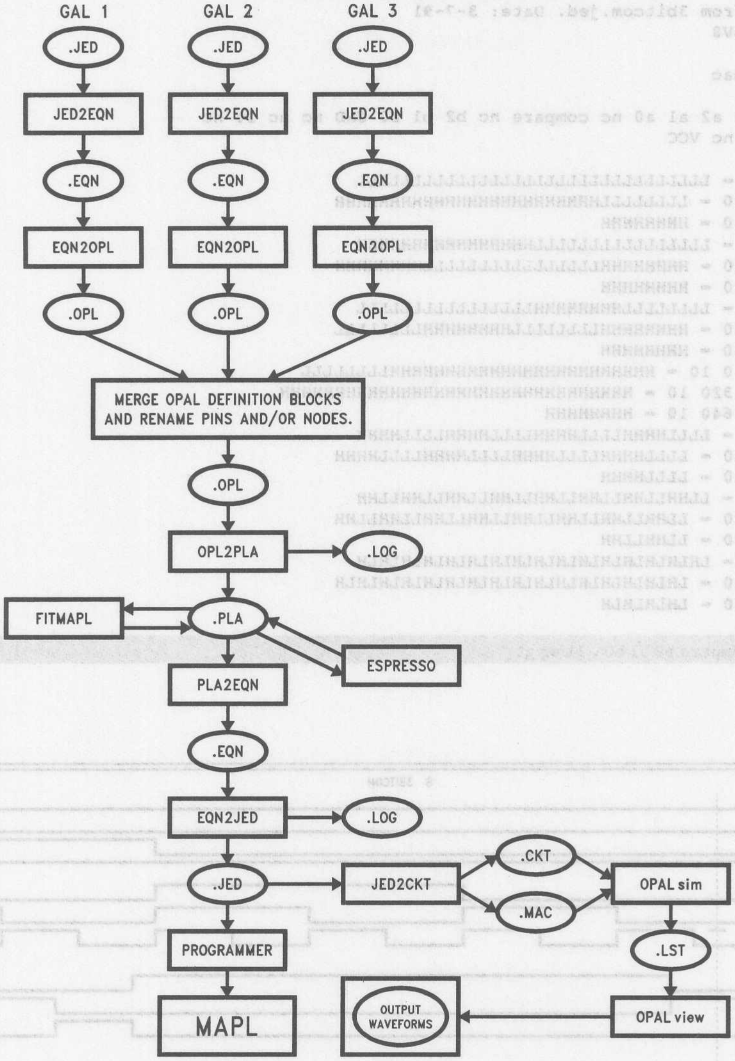


TL/L/9989-5

FIGURE 6. OPAL Waveform Simulation

Application Example 1

Multiple GAL to MAPL Conversion



TL/L/9989–7

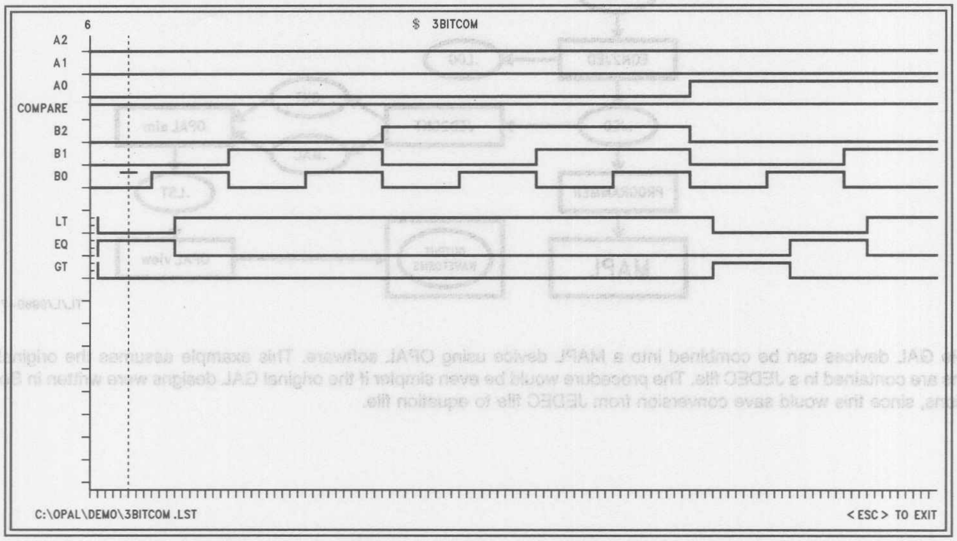
Multiple GAL devices can be combined into a MAPL device using OPAL software. This example assumes the original GAL designs are contained in a JEDEC file. The procedure would be even simpler if the original GAL designs were written in Boolean equations, since this would save conversion from JEDEC file to equation file.

Application Example 2

```
$ 3bitcom
$ JED2CKT - JEDEC File to DLSIM Circuit/Macro Translator (Version 1.00.B06)
$ Copyright (R) National Semiconductor Corporation 1990
$ Translated from 3bitcom.jed. Date: 3-7-91
$ DEVICE GAL16V8

.LIB 3bitcom.mac

* U1 3bitcom 2 a2 a1 a0 nc compare nc b2 b1 b0 GND nc nc lt nc
+ eq nc gt nc nc VCC

a2 INPUT 0 10 - LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
a2 INPUT 320 10 - LLLLLLLLHHHHHHHHHHHHHHHHHHHHHHHH
a2 INPUT 640 10 - HHHHHHHH
a1 INPUT 0 10 - LLLLLLLLLLLLLLLLHHHHHHHHHHHHHHHH
a1 INPUT 320 10 - HHHHHHHHLLLLLLLLLLLLLLLLHHHHHHHH
a1 INPUT 640 10 - HHHHHHHH
a0 INPUT 0 10 - LLLLLLLLHHHHHHHHLLLLLLLLLLLLLLLL
a0 INPUT 320 10 - HHHHHHHHLLLLLLLLHHHHHHHHLLLLLLLL
a0 INPUT 640 10 - HHHHHHHH
compare INPUT 0 10 - HHHHHHHHHHHHHHHHHHHHHHHHLLLLLLLL
compare INPUT 320 10 - HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
compare INPUT 640 10 - HHHHHHHH
b2 INPUT 0 10 - LLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHH
b2 INPUT 320 10 - LLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHH
b2 INPUT 640 10 - LLLLHHHH
b1 INPUT 0 10 - LLHHLLHHLLHHLLHHLLHHLLHHLLHHLLHH
b1 INPUT 320 10 - LLHHLLHHLLHHLLHHLLHHLLHHLLHHLLHH
b1 INPUT 640 10 - LLHHLLHH
b0 INPUT 0 10 - LHLHLHLHLHLHLHLHLHLHLHLHLHLHLHLH
b0 INPUT 320 10 - LHLHLHLHLHLHLHLHLHLHLHLHLHLHLHLH
b0 INPUT 640 10 - LHLHLHLH
```

TL/L/9989-8

```
.PROBE a2 a1 a0 compare b2 b1 b0 . lt eq gt

.TIME 719

.END
```



TL/L/9989-9

This example is taken from the 3BITBUS demonstration. This portion of the demonstration shows one of the many features of the interactive simulator, namely, the ability to arrange waveforms in a more meaningful organization by modifying the .CKT file. The modified portion appears here in larger text.

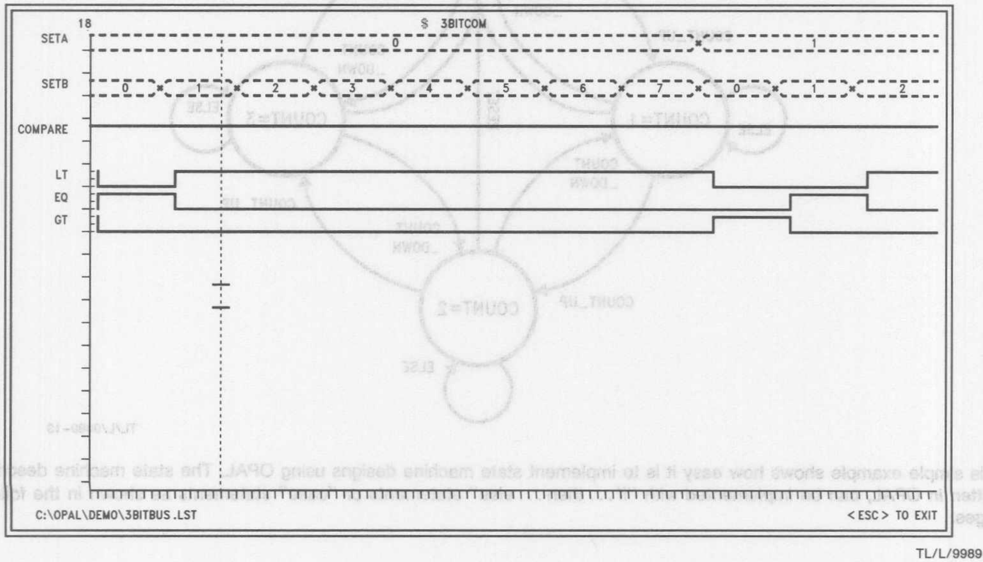
Application Example 2 (Continued)

```
$ 3bitcom
$ JED2CKT - JEDEC File to DLSIM Circuit/Macro Translator (Version 1.00.XXX)
$ Copyright (R) National Semiconductor Corporation 1990
$ Translated from 3bitcom.jed. Date: 2-26-91
$ DEVICE GAL16V8

.LIB 3bitcom.mac

* U1 3bitcom 2 a2 a1 a0 nc compare nc b2 b1 b0 GND nc nc lt nc
+ eq nc gt nc nc VCC

a2 INPUT 0 10 - LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
a2 INPUT 320 10 - LLLLLLLLHHHHHHHHHHHHHHHHHHHHHHHH
a2 INPUT 640 10 - HHHHHHHH
a1 INPUT 0 10 - LLLLLLLLLLLLLLLLHHHHHHHHHHHHHHHH
a1 INPUT 320 10 - HHHHHHHHLLLLLLLLLLLLLLLLHHHHHHHH
a1 INPUT 640 10 - HHHHHHHH
a0 INPUT 0 10 - LLLLLLLLHHHHHHHHLLLLLLLLLLLLLLLL
a0 INPUT 320 10 - HHHHHHHHLLLLLLLLHHHHHHHHLLLLLLLL
a0 INPUT 640 10 - HHHHHHHH
compare INPUT 0 10 - HHHHHHHHHHHHHHHHHHHHHHHHLLLLLLLL
compare INPUT 320 10 - HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
compare INPUT 640 10 - HHHHHHHH
b2 INPUT 0 10 - LLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHH
b2 INPUT 320 10 - LLLLHHHHLLLLHHHHLLLLHHHHLLLLHHHH
b2 INPUT 640 10 - LLLLHHHH
b1 INPUT 0 10 - LLHHLLHHLLHHLLHHLLHHLLHHLLHHLLHH
b1 INPUT 320 10 - LLHHLLHHLLHHLLHHLLHHLLHHLLHHLLHH
b1 INPUT 640 10 - LLHHLLHH
b0 INPUT 0 10 - LHLHLHLHLHLHLHLHLHLHLHLHLHLHLHLH
b0 INPUT 320 10 - LHLHLHLHLHLHLHLHLHLHLHLHLHLHLHLH
b0 INPUT 640 10 - LHLHLHLH
nc INPUT 0 - L
```

TL/L/9989-10

```
.bus seta a2 a1 a0
.bus setb b2 b1 b0
.PROBE seta . setb . compare . lt eq gt
```

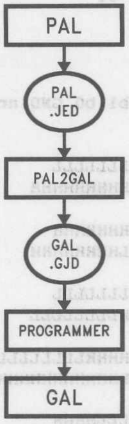
```
.TIME 719
.END
```



TL/L/9989-11

Application Example 3

**Design Upgrade:
PAL to GAL Conversion
with OPAL or OPALjr**

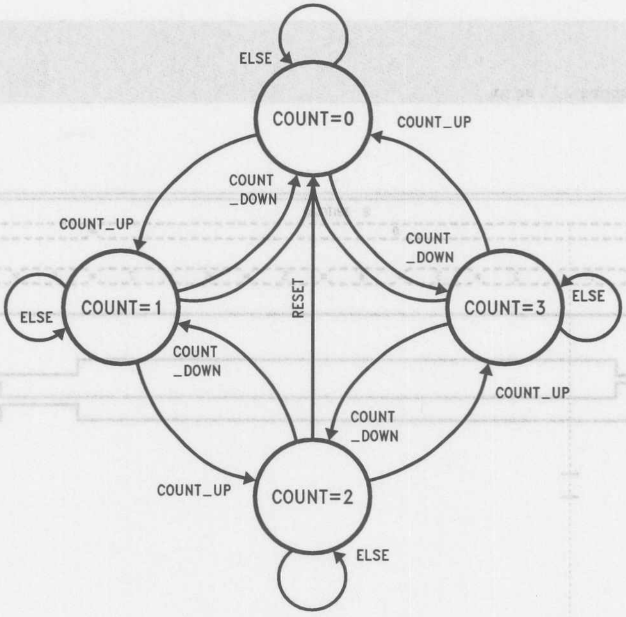


TL/L/9989-12

This example shows the conversion from PAL to GAL. Note that this procedure may be accomplished with either OPAL or OPALjr software packages.

Application Example 4

**Simple State Diagram:
Two-Bit Counter with Reset**



TL/L/9989-13

This simple example shows how easy it is to implement state machine designs using OPAL. The state machine description, written in OPAL, can be implemented with "if . . . then . . . else" statements or "case" statements as shown in the following pages.

Application Example 4 (Continued)

STATE MACHINE DESCRIPTION USING IF . . . THEN . . . ELSE

```
begin header
  This is a two-bit up-down counter with four decoded outputs.
  This counter will:
              count up if up is l and down is 0,
              count down if up is 0 and down is 1,
              and hold otherwise.
end header
begin definition
    inputs          clk,up, down, rst, /oe;
    outputs (com)   ol, o2, o3, 04;
    statebits       sb2, sbl;
    state_names     zero=0, one=^bl, two=^bl0, three=3;
    set             count = [sb2,sbl];
end definition
begin equation
    count.c = clk;
    count.oe = oe;
{
    count_up and count_down are not defined in the
    definition block, so they are intermediate variables.
}
    count_up = * /down;
    count_down = /up * down;
end equation
begin state_diagram count (sb2, sbl)
  state all  :
  if rst then zero;       ( If rst is high then reset counter. )
  state zero :
  ol = l;
  if count_up then one
  else if count_down then three
  else zero;
  state one :
  o2 = l;
  if count_up then two
  else if count_down then zero
  else one;
state two :
  o3 = l;
  if count_up then three
  else if count_down then one
  else two;
state three :
  o4 = l;
  if count_up then zero
  else if count_down then two
  else three;
end state_diagram
begin vector
. . . vectors . . .
end vector
```

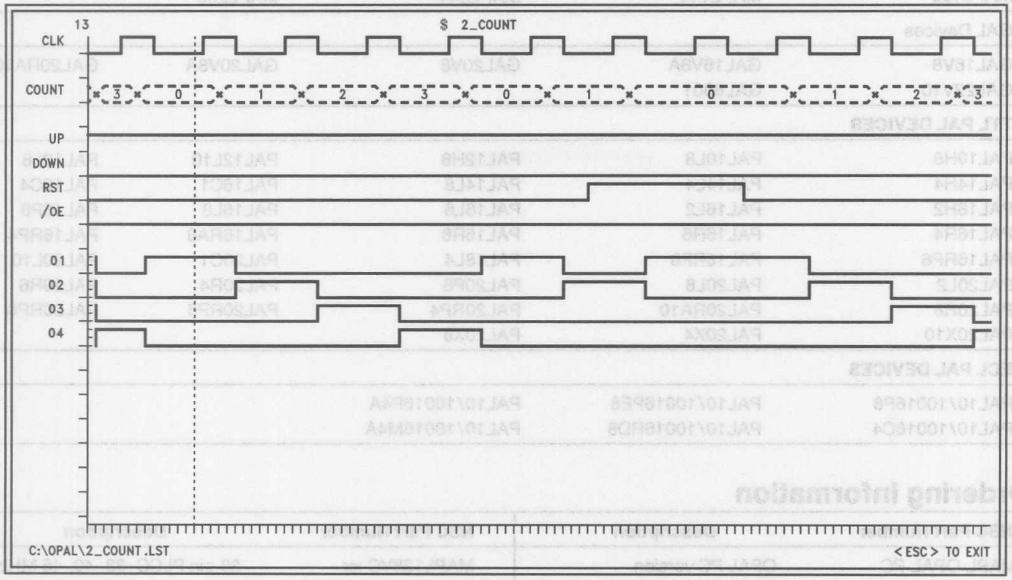
Application Example 4 (Continued)

STATE MACHINE DESCRIPTION USING CASE STATEMENTS

```
begin header
  This design is a 2-bit up/down counter with reset using CASE statements.
  It is to fit into one GAL16V8.
  It also illustrates the use of intermediate variables UP, DN and HLD.
end header
begin definition
  device             G16V8;
  inputs             clk=1,up=2,down=3,reset=4;
  statebits          sb1=15,sb0=16;
  state_names        zero=0,one=1,two=2,three=3;
  output ( com )     even=12,odd=13;
  set                cnt=[sb1,sb0]
end definition
begin equation
  even = /sb0;
  odd = sb0;
  UP = up * /down ;   ( Since UP, DN and HLD is not defined in the         )
  DN = /up * down;    ( definition block, they are treated as intermediate )
                      ( variables/pins.
  HLD = up !$ down ;  ( If up and down are both 0 or 1, counter will hold. )
end equation
begin state_diagram (sb1, sb0)     ( The left bit is the most significant bit. )
state all : if reset then zero;
state zero  : case UP     : one ;
                   DN     : three ;
                   HLD    : zero ;
              endcase ;
state one   : case UP     : two ;
                   DN     : zero ;
                   HLD    : one ;
              endcase ;
state two   : case UP     : three ;
                   DN     : one ;
                   HLD    : two ;
              endcase ;
state three : case UP     : zero ;
                   DN     : two ;
                   HLD    : three ;
              endcase ;
end state_diagram
begin vector
. . . vectors . . .
end vector
```

Application Example 4 (Continued)

Simulated Waveforms



TL/L/9989–14

This is the simulated waveform resulting from using either of the two previous OPAL input files.

Devices Supported

MAPL DEVICES				
MAPL128	MAPL144	MAPL244	MAPL268	
GAL Devices				
GAL16V8	GAL16V8A	GAL20V8	GAL20V8A	GAL20RA10
GAL22V10	GAL6001			
TTL PAL DEVICES				
PAL10H8	PAL10L8	PAL12H6	PAL12L10	PAL12L6
PAL14H4	PAL14L4	PAL14L8	PAL16C1	PAL16C4
PAL16H2	PAL16L2	PAL16L6	PAL16L8	PAL16P8
PAL16R4	PAL16R6	PAL16R8	PAL16RA8	PAL16RP4
PAL16RP6	PAL16RP8	PAL18L4	PAL20C1	PAL20L10
PAL20L2	PAL20L8	PAL20P8	PAL20R4	PAL20R6
PAL20R8	PAL20RA10	PAL20RP4	PAL20RP6	PAL20RP8
PAL20X10	PAL20X4	PAL20X8		
ECL PAL DEVICES				
PAL10/10016P8	PAL10/10016PE8	PAL10/10016P4A		
PAL10/10016C4	PAL10/10016RD8	PAL10/10016M4A		

Ordering Information

NSC Part Number	Description	NSC Part Number	Description
MAPL-OPAL-PC	OPAL PC version	MAPL128VC-xx	28-pin PLCC -33,-40,-45 MHz
MAPL-OPAL-PCW	OPAL PC Windows version	MAPL144VC-xx	44-pin PLCC -33,-40,-45 MHz
MAPL-OPAL-SUN4	OPAL SUN-4/UNIX	MAPL244VC-xx	44-pin PLCC -40, -45, -50 MHz
MAPL-OPAL-xxxx	Future platforms	MAPL268VC-xx	Future device

3rd Party Software

ABEL™ SOFTWARE

The ABEL package is an example of the second-generation PLD development tools which are usually referred to as high-level assemblers.

From an early version, which supported many PLD devices with logic reduction, simulation and generation of design documentation, it has become one of the standard software tools, capable of supporting a wide variety of PLDs, including PROMs. The most important feature which the most recent developments of ABEL software have added include:

- Revised simulation to support asynchronous devices and macrocells
- Syntax support of multiple-feedback paths
- Library of device-specific macros and functions
- JEDEC-to-ABEL conversion (for recovering undocumented designs)

ABEL software is available for platforms like the IBM/PC, VAX™ and others. The design can be entered using any standard text editor. Any combination of Boolean equations, truth tables or state diagrams can be used. The description falls into four main sections:

- Declarations section, where sets are defined
- Equations section, where Boolean equations are entered
- Truth Table section, where functional tables are entered
- Test Vector section, where the behavior during simulation is given

ABEL automatically performs logic reduction, simulation and conversion to a JEDEC file without requiring further intervention unless some error is encountered.

ATGEN SOFTWARE

ATGEN software automatically generates test vectors for CMOS, TTL, and ECL PLDs. The software generates effective tests for virtually all PLD designs including those that contain feedback, state machines, internal memory, and bidirectional I/O.

CUPL™ SOFTWARE

The CUPL package is a high-level compiler similar to ABEL software, which provides a number of functions not normally found in a standard PLD assembler. The input syntax is based on the C programming language. The high-level PLD support language in CUPL software permits development of designs using a systems approach. To this end, several features are supplied, including:

- self-documenting syntax
- state machine input option
- macro substitution
- flexible format
- use of symbolic names
- bit-field capability
- pre-processor functions
- output polarity selection

The macro substitution allows for considerable reduction in the number of keystrokes required for data entry, particularly for more complex functions. The pre-processor function allows the source file to be written in a much more generalized manner up until the actual compilation by the main assembler. This includes definition of the functions in state machine syntax and generalized arguments.

HITEST-PLD SOFTWARE

HITEST-PLD is an automatic test generation tool designed specifically to create high-fault coverage test waveforms for programmable logic devices (PLDs). HITEST-PLD produces high-quality test vectors that locate common manufacturing and customization defects in PLD components. HITEST-PLD's automatic capabilities eliminate manual test development and significantly reduce overall PLD test development time.

LOG/iC™ SOFTWARE

LOG/iC is a PLD synthesis tool featuring all the high level design entry formats required for easy and efficient designs. LOG/iC's HYPERPLD concept allows the user to first enter, simulate and process and design, without a need to pinpoint a specific device.

The proprietary PLD optimizer used for logic reduction is an exact PLD optimizer that is fast enough to be applicable for large designs. So, the actual minimal number of product terms is found, ensuring the most cost-effective hardware design.

After the reduction process, LOG/iC's PLD Data Base may be used to select the best device for the application. This selection process is based on the results of the HYPERPLD optimization as well as on additional interactive parameters like speed, power consumption, packaging, etc.

LOG/iC supports the mainline PALs, GALs as well as ECL devices. The library of supported parts is updated regularly. There is also a LOG/iC-GATES compiler available which accepts the same PLD design files assuring an easy growth path to gate array designs.

LOG/iC is available on a number of hosts like IBM-PC, SUN3, SUN4, SUN386, APOLLO, HP-9000 and VAX workstations/mainframes. On all hosts, LOG/iC is operated through the same menu driven interface with online help available. An integrated communications program eases the device programmer operation and the downloading of JEDEC files.

OrCAD SOFTWARE

OrCAD Programmable Logic Design Tools is one part of a fully integrated system which operates under OrCAD/ESP. Other OrCAD tool sets include: Schematic Design Tools, Digital Simulation Design Tools and PC Board Layout Tools. Programmable Logic Design Tools allows multiple design entry formats, including schematic capture, logic simulation, test vector generation, PLA file generation and device fitting.

PGADesigner SOFTWARE

PGADesigner are universal logic synthesis packages allowing the user to design logic functions using PLDs and FPGAs. PGADesigner allows waveform, schematic and high-level language entry and permits these entry formats to be used in combination. PGADesigner also features logic simulation, automatic device selection and partitioning, and downloading of the JEDEC file to a Data I/O programmer.

PLDlab90 SOFTWARE

PLDlab90 features automatic test vector generation, ambiguous timing delay simulation, design-for-test analysis and feedback and automatic documentation. PLDlab90 allows the testing of multiple PLDs (clusters) as they might appear on a board.

PLDtest Plus SOFTWARE

PLDtest combines a testability of a device with fault grading and automatic test vector generation. PLDtest will analyze the circuitry and determine the number of faults, how many of those faults can be tested, and will automatically generate test vectors to test those faults.

ViewPLD SOFTWARE

ViewPLD is a comprehensive PLD development package offering multiple design entry formats, automatic partitioning, device fitting, VHDL model generation, logic simulation, waveform analysis and symbol/schematic generation. Design entry is accomplished by ABEL input format or by importing JEDEC files. Viewdraw can be used to perform schematic capture, and the completed design can be automatically partitioned and minimized. Automatic device selection can be performed and device fitters are used to automatically make the design device specific. Logic simulation can be done either before or after the fitting stage using Viewwave and Viewsim/SD. A netlist and/or VHDL description can be generated for migrating designs to FPGAs and gate arrays.

For more information on the above packages contact:

ABEL:
Data I/O Corporation
10525 Willows Road NE
P.O. Box 97046
Redmond, WA 98073-9746
(206) 881-6444
1-800-247-5700

ATGEN:
ACUGEN Software, Inc.
427-3 Amherst Street, Suite 391
Nashua, NH 03063
(603) 891-1995

CUPL:
Logical Devices, Inc.:
1201 NW 65th Place
Ft. Lauderdale, Florida 33309
1-800-331-7766

HITEST-PLD
GenRad
510 Cottonwood Drive
Milpitas, CA 95035
(408) 432-1000
FAX: (408) 943-8240 or 432-1890

LOG/IC:
800 Airport Road
Monterey, CA 93940
(408) 373-7359

OrCAD:
OrCAD
3175 NW Aloclek Drive
Hillsboro, OR 97124-7135
Sales and Administration: (503) 690-9881
24-Hour Bulletin Board System: (503) 690-9791
FAX: (503) 690-9891

PGADesigner:
Minc Incorporated
6755 Earl Drive
Colorado Springs, CO 80918
(719) 590-1155
FAX: (719) 590-7330

PLDlab90:
Informations- und Nachrichtentechnik
iNt GmbH Bunsenstrasse 6
D-8033 Martinsried
089 857 66 67
FAX: 089 856 12 13

PLDtest Plus:
Data I/O Corporation
10525 Willows Road NE
P.O. Box 97046
Redmond, WA 98073-9746
(206) 881-6444
1-800-247-5700

ViewPLD:
Viewlogic Systems, Inc.
293 Boston Post Road W
Marlboro, MA 01752
(508) 480-0881
1-800-422-4660
FAX: (508) 480-0882

PLD Programming and Testing

Many programming and test issues exist today that were not issues a short time ago, due to the increased speed, density, and complexity of many of today's devices. Some of these issues include: inadequate hardware pin drivers that cause double clocking and glitches, poor V_{CC} and ground planes that cause ground bounce, ESD sensitive CMOS devices, and improper use of test vectors.

PROGRAMMING

The use of NSC-approved programming equipment will ensure a quality programmed device. Approved programmers have undergone extensive evaluation by NSC and must meet NSC's test criteria in a number of programming related areas. Use of non-certified programming equipment (such as IC testers with custom algorithms) may result in improperly programmed devices. National Semiconductor is continuously evaluating new programming equipment and handlers; please contact a National Semiconductor representative for the latest list of approved equipment.

QUALITY IN PROGRAMMING (QIP) CENTERS

National Semiconductor has established a worldwide network of distributor QIP centers to guarantee the highest degree of quality and service for programming and testing of PLD's. The QIP centers are certified and approved by NSC after having met stringent requirements including; use of approved programming and test equipment, programming and handling procedures, personnel training and ESD protection.

Being locally situated, our QIP centers are able to meet all of your programming requirements, and guarantee the output quality. For more information contact your local National Semiconductor sales representative.

TESTING

Many test installations that have been used successfully in the past are no longer adequate for testing today's PLDs. For example: PLD device programmers have routinely been used as a functional tester for reliability and quality screening. This is no longer a valid use of programmers. Analysis has shown that most of today's programmers are incapable of reliably testing high-speed or complex PLDs because most programmers are not specifically designed to perform accurate testing. Doing so would mean having to use costly hardware which makes the programmer very expensive.

HANDLERS AND IC TESTERS

When automatic device handlers are used, additional programming and test issues exist that the customer must be aware of. Some of these include: increased ESD environments, the inability of pick-and-place equipment to properly handle new packages and degraded program and test signals at the handler contacts. Automated environments should be carefully evaluated to ensure high quality programming/testing and high yields.

Even large commercial IC test systems are affected by these issues, especially when device programming is part of an automatic flow. The programming algorithms implemented on these systems are usually highly customized and may violate the NSC programming specifications.

OPTIONS

The end users of high-speed or complex PLDs, especially users of small quantities, may choose not to test these devices at all, relying on the quality of the devices and of the programmers used to program them. Many programmers reliably program devices, but do not reliably functionally test these same devices.

End users that use large volumes of a PLD must look beyond device programmers for reliable functional testing. Stand-alone test boxes or more complex IC testers must be used.

When using more complex setups perform multiple device insertions and device labeling, however, extra care must be taken to ensure that reliable, high-yield results are obtained.

TABLE 1. Programmer Manufacturers Supporting National PLDs

Manufacturer	Model
Advantech	
Taiwan: Shing-Tien City, Taipei 886-2-9184567, FAX: 886-2-9184566	PC-UPROG
England: Dataman 44-305-68066, FAX: 44-305-64997	OMNIPRO-II
USA: San José, CA (408) 293-6786, FAX: (408) 293-4697	
American Reliance (818) 575-5110	AR9860
ADVIN	SAILOR-PAL
USA: Sunnyvale, CA (408) 984-8600, FAX: (408) 736-2503	PILOT
AVAL Corporation	PKW-3100
Ireland: Deansgrange, Co. Dublin 353-1-892136, FAX: 353-1-892070	PKW-5100
Japan: Tokyo 03-344-2001, FAX: 03-344-2007	
USA: El Segundo, CA Electramotive (213) 722-9208, FAX: (213) 322-1716	
BP Microsystems	PLD-1100
USA: Houston, TX (713) 461-9430, FAX: (713) 461-4713	CP/PLD-1128
Great Britain: Mutek, Ltd. 2216-6501, FAX: 2216-5083	
Germany: API Electronik 8136-7092, FAX: 8136-7398	
France: Emulations 1-69-41-2801, FAX: 1-60-19-2950	
Data I/O Corporation	29B
USA: Redmond, WA (206) 881-6444, FAX: (206) 882-1043	60H 2900 3900
Canada: Mississauga, Ontario (416) 678-0761	UNISITE
Europe: Amsterdam +31-0-20-6622866	
Japan: Tokyo 03-432-6991	
Digelec	860
USA: West Hills, CA (818) 887-3755, FAX: (818) 887-3693	
Germany: Munich 089-776-098, FAX: 089-725-9164	
GP Industrial	AP100
England: Plymouth 0752-342961, TELEX: 42513 SHARET G	
HI-LO Systems	
Taiwan: Taipei 02-7640215, FAX: 886-2-7566403	ALL-03
USA: Tribal Microsystems (415) 623-8860, FAX: (415) 623-9925	TUP-300
Logical Devices	PALPRO
USA: Ft. Lauderdale, FL (305) 974-0975, FAX: (305) 974-8531	ALLPRO
Minato Electronics	1890A
Japan: Yokohama 045-591-5611, FAX: 045-591-5618	
USA: North Highlands, CA (916) 348-6066, FAX: (916) 348-0926	
SMS GmbH	SPRINT PLUS
Germany: Hergatz 7522-5018, FAX: 7522-8929	SPRINT EXPERT
USA: Redmond, WA 98052 (206) 883-8447, FAX: (206) 883-8601	
Stag Microsystems	ZL30A/B
England: Welwyn Garden City, Herts 0707-332148, FAX: 707-371503	SYSTEM 3000
USA: Santa Clara, CA (408) 988-1118	
Canada: Mississauga, Ontario (416) 890-2010	
System General	SGUP = 85A
Taiwan: Taipei 886-2-917-3005, FAX: 886-2-911-1283	TURPRO = 1
USA: Milpitas, CA (408) 263-6667, FAX: (408) 262-9220	

Section 5

Appendices/ Physical Dimensions

Section 5 Contents

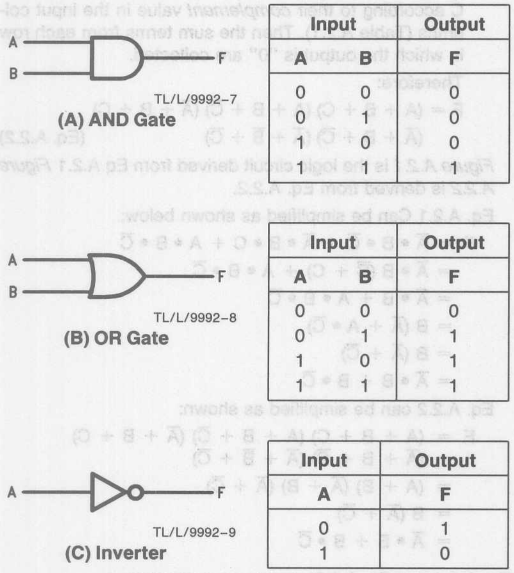
Appendix A—Boolean Logic Review 5-3
A.1 Basic Operators and Theorems 5-3
A.2 Derivation of a Boolean Expression 5-4
A.3 Minimization 5-6
A.4 K-Map Method 5-7
A.5 Sequential Circuit Elements 5-11
A.6 State Machine Fundamentals 5-12
A.7 Avoiding Logic Hazards 5-14
Appendix B—Theory of PLD Testing 5-16
B.1 Testing Methods 5-16
B.2 Test Generation Techniques 5-17
B.3 Response Evaluation Techniques 5-27
Physical Dimensions 5-32
Bookshelf
Distributors

National Semiconductor

Appendix A
Boolean Logic Review

A.1 Basic Operators and Theorems

A gate is an electronic circuit which operates on one or more input signals to produce an output signal. There are three basic gates from which all other logic can be realized: AND, OR, and INVERTER gates. *Figure A.1.1* shows these three basic gates and their truth table.



(A) AND Gate

Input		Output
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

(B) OR Gate

Input		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

(C) Inverter

Input	Output
A	F
0	1
1	0

FIGURE A.1.1. Basic Gates

To express the function of these gates by Boolean algebra, we need to define Boolean operators as follows:

=	Logical Equality
$\overline{x}$	Negate (Not, Invert, Complement)
+	OR (Sum)
•	AND (Product)
⊕	Exclusive OR

The function of an AND gate in *Figure A.1.1* can be expressed as:

$$F = A \bullet B$$

The function of an OR gate and INVERTER can be expressed as:

$$F = A + B$$

and $F = \overline{A}$

Boolean operators are logical operators, which are different from arithmetic operators. For example, + is logical addition, • is logical multiplication. We call such equations Boolean equations or logic equations.

A number of logic theorems and laws will be used to manipulate and reduce logical equations. These theorems and laws are as follows:

Theorem 1	$A + 0$	$= A$
Theorem 2	$A \bullet 0$	$= 0$
Theorem 3	$A + 1$	$= 1$
Theorem 4	$A \bullet 1$	$= A$
Theorem 5	$A + A$	$= A$
Theorem 6	$A \bullet A$	$= A$
Theorem 7	$A + \overline{A}$	$= 1$
Theorem 8	$A \bullet \overline{A}$	$= 0$
Theorem 9	$\overline{\overline{A}}$	$= A$
Theorem 10	$A + A \bullet B$	$= A$
Theorem 11	$A \bullet (A + B)$	$= A$
Theorem 12	$(A + B) \bullet (A + C)$	$= A + B \bullet C$
Theorem 13	$A + \overline{A} \bullet B$	$= A + B$

Commutative Law

$$A + B = B + A$$
$$A \bullet B = B \bullet A$$

Associative Law

$$A + B + C = (A + B) + C = A + (B + C)$$
$$A \bullet B \bullet C = (A \bullet B) \bullet C = A \bullet (B \bullet C)$$

Distributive Law

$$A + (B \bullet C \bullet D) = (A + B) \bullet (A + C) \bullet (A + D)$$
$$A \bullet (B + C + D) = A \bullet B + A \bullet C + A \bullet D$$

DeMorgan's Theorem

$$\overline{(A + B + C)} = \overline{A} \bullet \overline{B} \bullet \overline{C}$$
$$\overline{(A \bullet B \bullet C)} = \overline{A} + \overline{B} + \overline{C}$$

The complement of any Boolean expression, or a part of any expression, may be found by means of DeMorgan's theorem. Two steps are used to form a complement in this theorem:

1. OR symbols are replaced with AND symbols or AND symbols with OR symbols.
2. Each of the terms in the expression is complemented.

DeMorgan's theorem is one of the most powerful tools for engineering applications. It is very useful for designing with programmable logic devices because it provides a quick and simple conversion method between PRODUCT-OF-SUMS and SUM-OF-PRODUCTS expressions, which will be defined later.

A.2 Derivation of a Boolean Expression

Any logic expression can be reduced to a two-level form and expressed as either a SUM-OF-PRODUCTS (SOP) or PRODUCT-OF-SUMS (POS). Before we define SOP or POS, we need to define "terms".

1. **Product Term:** A product term is a single variable or the logical product of several variables. The variable may or may not be complemented.
2. **Sum Term:** A sum term is a single variable or the sum of several variables. The variables may or may not be complemented.
3. **Normal Term:** A normal term is a product or sum term in which no variable appears more than once.
4. **Minterm:** A minterm is a product term containing every variable once and only once (either true or complemented).
5. **Maxterm:** A maxterm is a sum term containing every variable once and only once (either true or complemented).

For example, the term $A \bullet B \bullet C$ is a product term; $A + B$ is a sum term; A is both a product term and a sum term; $A + B \bullet C$ is neither a product term nor a sum term; $A + \overline{B}$ is a sum term; $A \bullet \overline{B} \bullet \overline{C}$ is a product term; $\overline{B}$ is both a sum term and a product term. We now define two most important forms:

1. **SUM-OF-PRODUCTS Expression:** A sum-of-products expression is a product term or several product terms logically added together.
2. **PRODUCT-OF-SUMS Expression:** A product-of-sums expression is a sum term or several sum terms logically multiplied together.

For example, the expression $\overline{A} \bullet B + A \bullet \overline{B}$ is a sum-of-products expression; $(A + B) \bullet (\overline{A} + \overline{B})$ is a product-of-sums expression.

One prime reason for using sum-of-products or product-of-sums expressions is their straightforward conversion to very simple gating networks. In their purest, simplest form they go into two-level networks, which are networks for which the longest path through which a signal must pass from input to output is two gates long.

When designing a logic circuit, the logic designer works from two sets of known values; the various states which the inputs to the logical network can take, and the desired outputs for each input condition. The logic expression is derived from these sets of values and the procedure is as follows:

1. Construct a table of the input and output values (Table A.2.1 left half).

2a. To derive a SUM-OF-PRODUCTS (SOP) expression:

A product term column is added listing the inputs A, B, and C according to their value in the input columns (Table A.2.1). Then the product terms from each row in which the output is a "1" are collected.

Therefore:

$$F = \overline{A} \bullet B \bullet \overline{C} + \overline{A} \bullet B \bullet C + A \bullet B \bullet \overline{C} \qquad \text{(Eq. A.2.1)}$$

2b. To derive a PRODUCT-OF-SUMS (POS) expression:

A sum term column is added listing the inputs A, B, and C according to their *complement* value in the input columns (Table A.2.1). Then the sum terms from each row in which the output is "0" are collected.

Therefore:

$$F = (A + B + C)\,(A + B + \overline{C})\,(\overline{A} + B + C)\,(\overline{A} + B + \overline{C})\,(\overline{A} + \overline{B} + \overline{C}) \qquad \text{(Eq. A.2.2)}$$

Figure A.2.1 is the logic circuit derived from Eq A.2.1 *Figure A.2.2* is derived from Eq. A.2.2.

Eq. A.2.1 Can be simplified as shown below:

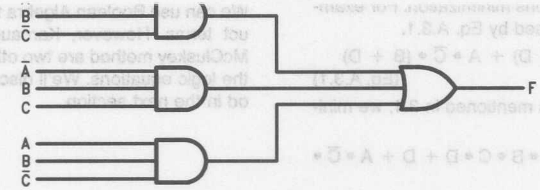
$$\begin{aligned} F &= \overline{A} \bullet B \bullet \overline{C} + \overline{A} \bullet B \bullet C + A \bullet B \bullet \overline{C} \\ &= \overline{A} \bullet B\,(\overline{C} + C) + A \bullet B \bullet \overline{C} \\ &= \overline{A} \bullet B + A \bullet B \bullet \overline{C} \\ &= B\,(\overline{A} + A \bullet \overline{C}) \\ &= B\,(\overline{A} + \overline{C}) \\ &= \overline{A} \bullet B + B \bullet \overline{C} \end{aligned}$$

Eq. A.2.2 can be simplified as shown:

$$\begin{aligned} F &= (A + B + C)\,(A + B + \overline{C})\,(\overline{A} + B + C) \\ &\quad (\overline{A} + B + \overline{C})\,(\overline{A} + \overline{B} + \overline{C}) \\ &= (A + B)\,(\overline{A} + B)\,(\overline{A} + \overline{C}) \\ &= B\,(\overline{A} + \overline{C}) \\ &= \overline{A} \bullet B + B \bullet \overline{C} \end{aligned}$$

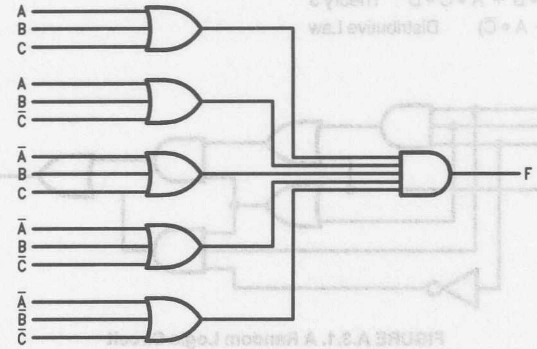
TABLE A.2.1. Truth Table Eq. A.2.1 and Eq. A.2.2

Inputs			Outputs	Product Terms	Sum Terms
A	B	C	F		
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	$A + B + C$
0	0	1	0	$\overline{A}\,\overline{B}\,C$	$A + B + \overline{C}$
0	1	0	1	$\overline{A}\,B\,\overline{C}$	$A + \overline{B} + C$
0	1	1	1	$\overline{A}\,B\,C$	$A + \overline{B} + \overline{C}$
1	0	0	0	$A\,\overline{B}\,\overline{C}$	$\overline{A} + B + C$
1	0	1	0	$A\,\overline{B}\,C$	$\overline{A} + B + \overline{C}$
1	1	0	1	$A\,B\,\overline{C}$	$\overline{A} + \overline{B} + C$
1	1	1	1	$A\,B\,C$	$\overline{A} + \overline{B} + \overline{C}$



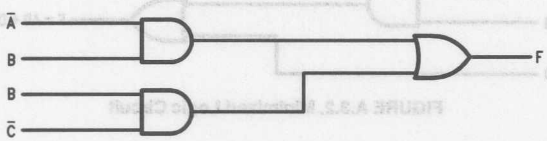
TL/L/9992–10

FIGURE A.2.1. Logic Circuits of Eq. A.2.1



TL/L/9992–11

FIGURE A.2.2. Logic Circuits of Eq. A.2.2



TL/L/9992–12

FIGURE A.2.3. Simplified Logic Circuits

A.3 Minimization

Logic circuits can be represented by logic expressions or so called logic equations. As discussed, we can minimize the logic circuit through logic equations minimization. For example, *Figure A.3.1* can be expressed by Eq. A.3.1.

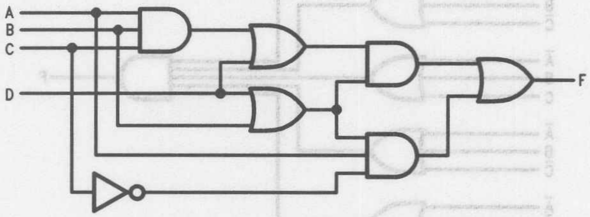
$$F = (A \bullet B \bullet C + D) \bullet (B + D) + A \bullet \overline{C} \bullet (B + D) \qquad \text{(Eq. A.3.1)}$$

By using the theorems and laws mentioned in 3.1, we minimize Eq. A.3.1 as follows:

$$\begin{aligned} F &= A \bullet B \bullet C + B \bullet D + A \bullet B \bullet C \bullet D + D + A \bullet \overline{C} \bullet B + A \bullet \overline{C} \bullet D \\ &= A \bullet B \bullet C\,(1 + D) + D(B + 1) + A \bullet \overline{C} \bullet B + A \bullet \overline{C} \bullet D && \text{Distributive Law} \\ &= A \bullet B \bullet C + D + A \bullet \overline{C} \bullet B + A \bullet \overline{C} \bullet D && \text{Theory 3} \\ &= A \bullet B\,(C + \overline{C}) + D\,(1 + A \bullet \overline{C}) && \text{Distributive Law} \\ &= A \bullet B + D \end{aligned}$$

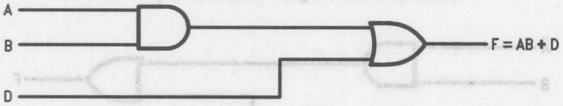
The minimum SOP expression can now be implemented as the simple AND-OR logic circuits as shown in *Figure A.3.2.*

We can use Boolean Algebra to reduce the number of product terms. However, Karnaugh Mapping and the Quine-McCluskey method are two other powerful tools to minimize the logic equations. We'll discuss Karnaugh Mapping method in the next section.



TL/L/9992-13

FIGURE A.3.1. A Random Logic Circuit



TL/L/9992-14

FIGURE A.3.2. Minimized Logic Circuit

A.4 K-Map Method

A Karnaugh map, hereafter called a K-map, is a graphical method for representing a Boolean function. It is similar to a truth table in that the K-map supplies the TRUE or FALSE value of a Boolean function for all possible combinations of its logical argument. There are many ways in which a K-map can be arranged. The most important considerations of the arrangement are:

1. There must be a unique location on the K-map for entering the TRUE/FALSE value of the function that corresponds to each combination of input variables.
2. The locations should be arranged so, with minimization mentioned in Section A.3, that they are readily apparent to the trained observer.

The second consideration implies that a successful K-mapping arrangement should point to groups of minterms or maxterms that can be combined into reduced forms. K-maps are also useful in expanding partially reduced expressions into standard forms prior to the minimization process.

The K-map is one of the most powerful tools at the hands of the logic designer. The power of the K-map does not lie in its application of any marvelous new theorems, but rather in its utilization of the remarkable ability of the human mind to perceive patterns in pictorial representations of data. This is not a new idea. Anytime we use a graph instead of a table of numerical data, we are utilizing the human ability to recognize complex patterns and relationships in a graphical representation far more rapidly and surely than in a tabular representation. A few examples of how to create a K-map follow.

First, consider a truth table for two variables. We list all four possible input combinations and the corresponding function values, i.e., the truth tables for AND and OR. *(Figure A.4.1)*

A	B	A • B
0	0	0
0	1	0
1	1	1
1	0	0

A	B	A + B
0	0	0
0	1	1
1	1	1
1	0	1

FIGURE A.4.1. Truth Tables for AND and OR

As an alternative approach, set up a diagram consisting of four small boxes, one for each combination of variables. Place a "1" in any box representing a combination of variables for which the function has the value 1. There is no logical objection to putting "0's" in the other boxes, but they are usually omitted for clarity.

The diagrams in *Figure A.4.2(a)* are perfectly valid K-maps, but it is more common to arrange the four boxes in a square, as shown in *Figure A.4.2(b).*

Since there must be one square for each input combination, there must be 2^n squares in a K-map for n-variables. Whatever the number of variables, we may interpret the map in terms of a graphical form of the truth table *(Figure A.4.3(a))* or in terms of union and intersection of areas *(Figure A.4.3(b))*. The K-maps for some other three-variable functions are shown in *Figure A.4.4.*

Particularly note the functions mapped in *Figure A.4.3(a)* and *A.4.4(b)*. These are both minterms. Each is represented by one square, obviously, and each one of the eight squares corresponds to one of the eight minterms of three variables. This is the origin of the name minterm. A minterm is the form of Boolean function corresponding to the minimum possible area, other than 0, on a K-map. A maxterm, on the other hand, is the form of Boolean function corresponding to the maximum possible area, other than 1, on a K-map. *Figure A.4.3(b)* and *A.4.4(c)* are two examples.

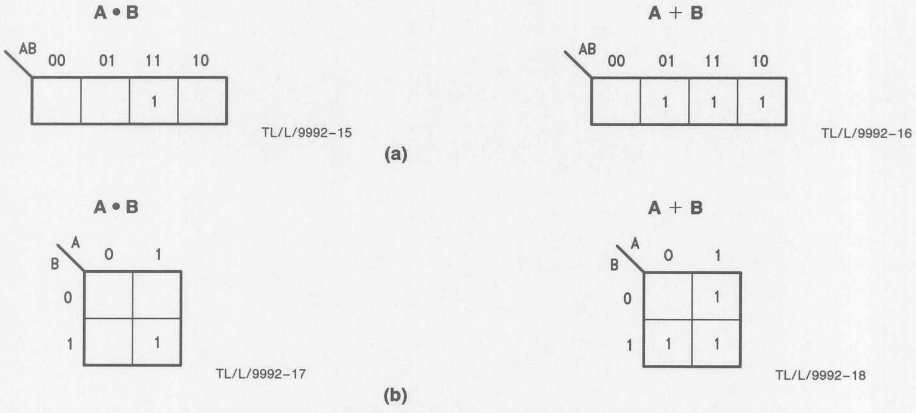
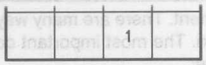
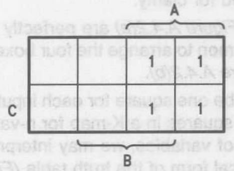


FIGURE A.4.2. K-Maps for AND and OR

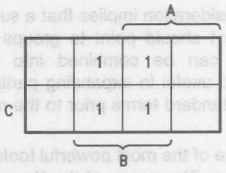

TL/L/9992-19

0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

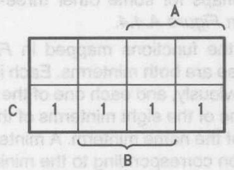
(a)



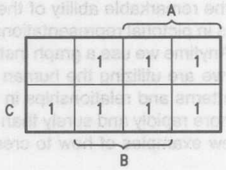
TL/L/9992-20



TL/L/9992-21



TL/L/9992-22

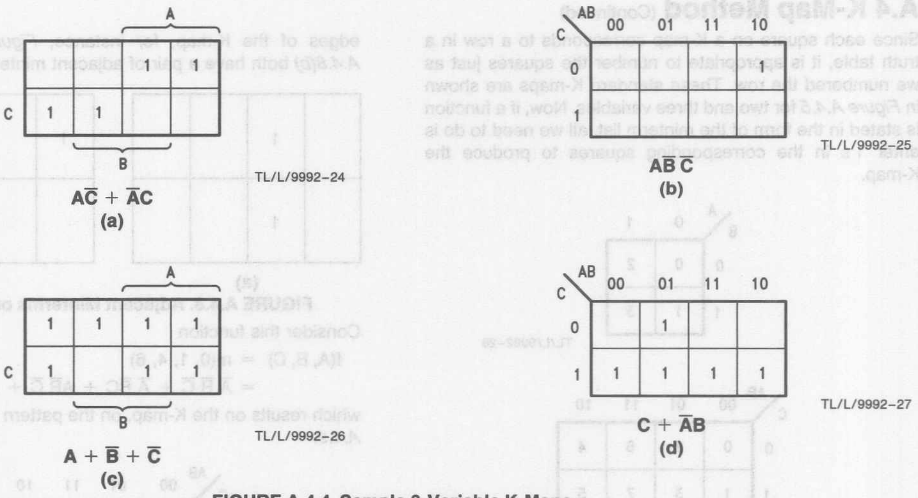


TL/L/9992-23

$$A + B + C = A + B + C$$

(b)

FIGURE A.4.3. K-Maps for 3-Variable AND and OR



$A\bar{C} + \bar{A}C$

(a)

TL/L/9992-24

$A\bar{B}C$

(b)

TL/L/9992-25

$A + \bar{B} + \bar{C}$

(c)

TL/L/9992-26

$C + \bar{A}B$

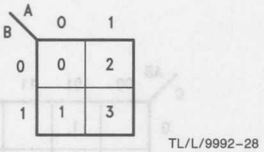
(d)

TL/L/9992-27

FIGURE A.4.4. Sample 3-Variable K-Maps

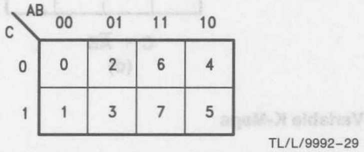
A.4 K-Map Method (Continued)

Since each square on a K-map corresponds to a row in a truth table, it is appropriate to number the squares just as we numbered the row. These standard K-maps are shown in *Figure A.4.5* for two and three variables. Now, if a function is stated in the form of the minterm list, all we need to do is enter 1's in the corresponding squares to produce the K-map.



A B	0	1
0	0	2
1	1	3

TL/L/9992-28



AB C	00	01	11	10
0	0	2	6	4
1	1	3	7	5

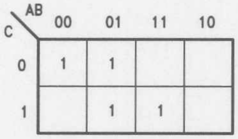
TL/L/9992-29

FIGURE A.4.5. K-Maps for Two and Three Variables

If a function is stated as a maxterm list, we can enter 0's in the squares listed or 1's in those not listed.

A map showing the 0's of a function is a perfectly valid K-map, although it is more common to show the 1's.

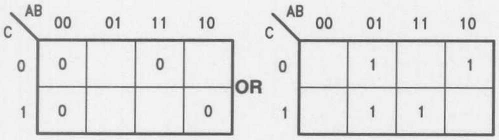
For example, the K-map of f(A, B, C) = m(0, 2, 3, 7) is shown in *Figure A.4.6* and the K-map of f(A, B, C) = M(0, 1, 5, 6) is shown in *Figure A.4.7* where m means minterm, M means maxterm.



AB C	00	01	11	10
0	1	1		
1		1	1	

TL/L/9992-30

FIGURE A.4.6. K-Map of M(0, 2, 3, 7)



AB C	00	01	11	10
0	0		0	
1	0			0

OR

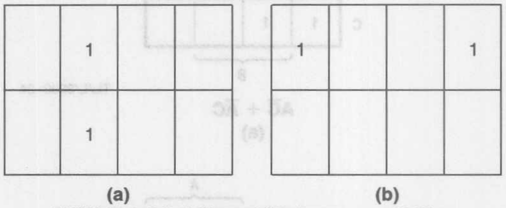
AB C	00	01	11	10
0		1		1
1		1	1	

TL/L/9992-31 TL/L/9992-32

FIGURE A.4.7. K-Map of M(0, 1, 5, 6)

As shown, the K-map can be generated from the truth table on minterm expression or maxterm expression. For the remainder of this section, we will learn how to minimize the minterm expression by using the K-map.

The general principle of this minimization technique is "Any pair of n-variable minterms which are adjacent on a K-map may be combined into a single product term of n − 1 literals." The definition of "adjacent" should include opposite edges of the K-map, for instance, *Figure A.4.8(a)* and *A.4.8(b)* both have a pair of adjacent minterms.



	1		
	1		

(a)

1			1

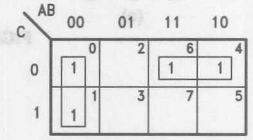
(b)

FIGURE A.4.8. Adjacent Minterms on a K-Map

Consider this function

$$f(A, B, C) = m(0, 1, 4, 6)$$
$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + AB\overline{C}$$

which results on the K-map, on the pattern shown in *Figure A.4.9.*



AB C	00	01	11	10
0	1		1	1
1	1			

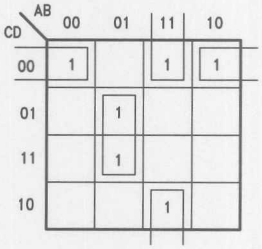
TL/L/9992-33

FIGURE A.4.9. Minimization

Therefore, combine minterms 0 and 1, 4 and 6 to get a minimal expression:

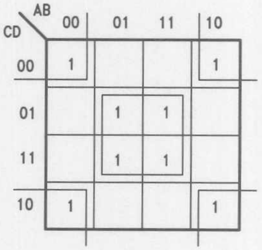
$$f(A, B, C) = \overline{A}\,\overline{B} + A\overline{C}$$

Figure A.4.10 shows some examples. Notice that it is permissible to include a minterm in several terms if it helps make the term shorter.



AB CD	00	01	11	10
00	1		1	1
01		1		
11		1		
10			1	

TL/L/9992-34

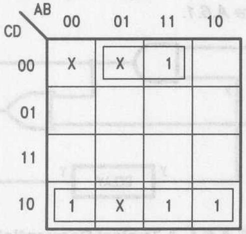


AB CD	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10	1			1

TL/L/9992-35

FIGURE A.4.10. Minimization

K-map makes it easy to take advantage of these "don't care" conditions by letting the "don't care" minterms be 1 or 0, depending on which value results in a simpler expression. *Figure A.4.11* shows an example of the use of "don't cares" (redundancies) to simplify the terms.



CD \ AB	00	01	11	10
00	X	X	1	
01				
11				
10	1	X	1	1

TL/L/9992–36

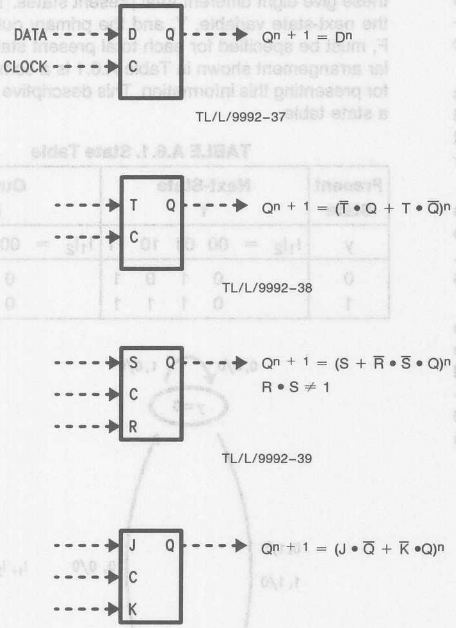
FIGURE A.4.11. Minimization

When working with larger functions, the tabular reduction developed by Quine and modified by McCluskey is an alter- expression for a Boolean functions to all other minterms with which it may form a combinable grouping.

The reader can refer to "Introduction to Switching Theory and Logic Design" by Hill and Peterson to understand the Quine-McCluskey method.

A.5 Sequential Circuit Elements

Usually the subject of logic design is subdivided into two types: sequential and combinational. A purely combinational logic subsystem has no memory. Its outputs are completely defined by its present inputs. The analysis and design of combinational logic is much easier. A sequential logic subsystem has memory and its outputs are functions of not only present inputs but the previous outputs. Circuits of multiplexer/selector, decoder/encoder, adder, and comparator are examples of combinational circuits. Shift register, counter, state machine, and memory controller are examples of sequential circuits.


DATA
CLOCK
D
Q
C
$Q^{n+1} = D^n$
T
Q
C
$Q^{n+1} = (\overline{T} \bullet Q + T \bullet \overline{Q})^n$
S
Q
C
R
$Q^{n+1} = (S + \overline{R} \bullet \overline{S} \bullet Q)^n$
$R \bullet S \neq 1$
J
Q
C
K
$Q^{n+1} = (J \bullet \overline{Q} + \overline{K} \bullet Q)^n$

TL/L/9992–37

TL/L/9992–38

TL/L/9992–39

TL/L/9992–40

D^n	Q^{n+1}
0	0
1	1

T^n	Q^{n+1}
0	Q^n
1	$(\overline{Q})^n$

R	S	Q^{n+1}
0	0	Q^n
0	1	1
1	0	0
1	1	X

J	K	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	$(\overline{Q})^n$

FIGURE A.5.1. Basic Flip-Flops

Just as we have a logic gate as the basic combinational circuit element, we have a flip-flop as a basic sequential circuit element. A flip-flop is a memory device which can remember, or store, a binary bit of information. There are four basic flip-flop types: (1) D flip-flop, (2) T flip-flop, (3) RS flip-flop, and (4) JK flip-flop. *Figure A.5.1* shows these elements and their truth table.

With the memory elements, the output does not change as a function of the inputs until the clock transition. Therefore, a superscript notation is used to indicate that the output during clock period n + 1 is a function of the inputs during the previous clock period n.

The D (delay) flip-flop means the input (D) is "stored" in the flip-flop when the clock occurs and will appear on the output (Q) during the next (n + 1) clock time. The D flip-flop is thus very much like a single-bit RAM. It is very useful for data storage and other special applications.

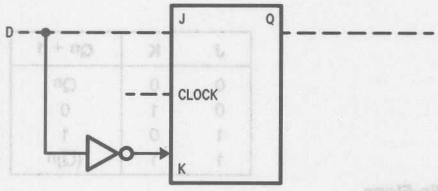
The other three types of flip-flops defined in *Figure A.5.1* are also one-bit storage elements, but instead of simply storing the input, they change state in response to the inputs by various logical rules. Since they hold their previous state in spite of the clock, unless an input goes true, they often simplify the combinational logic functions required to control them in control applications.

The T (toggle) flip-flop, for example, stays in its previous state if the T input is false before the clock. If the T input is true, the output changes to the opposite state (toggle) on the clock. The T flip-flop is thus useful, for example, in binary counters where we want each bit to invert every time there is a carry from the lower order bits.

The R–S flip-flop sets after the S input is true and resets after the R input is true. Its output is undefined if both R and S are true. It is possible to define a Set Overrides Reset (SOR) or a Reset Overrides Set (ROS) flip-flop. It will set or reset respectively if both the R and the S inputs are true.

The J–K flip-flop sets after J is true and resets after K is true. It is similar to an R–S flip-flop except that if J and K are both true, the output changes to the opposite state (toggle). It can be used as a T flip-flop by tying the J and K inputs together.

Since the J–K flip-flop can essentially do the job of both the R–S and the T flip-flop, the R-S and the T flip-flops are seldom seen. The choice is between J–K flip-flops for small counters and control or D flip-flops for data storage applications. Actually the J–K flip-flop can even do the job of the D flip-flop with the addition of a single inverter, as shown in *Figure A.5.2*.



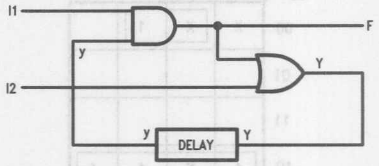
TL/L/9992–41

FIGURE A.5.2. Implement D Flip-Flop by Using J–K

Another memory element type, called a latch, is often described on data sheets with a truth table like the one for the D flip-flop in *Figure A.5.1*. It is definitely not like a D flip-flop, however, because the output changes as soon as the clock goes high and does not "latch" until the clock falls (if the input changes while the clock is high, the output follows it). Because of this characteristic, a latch is not usable in the synchronous logic.

A.6 State Machine Fundamentals

The relationships among present-state variables, primary input variables, next-state (or excitation) variables, and primary output variables that describe the behaviour of a sequential system can be specified in several ways. As an example, consider the simple sequential system that is shown in *Figure A.6.1*.



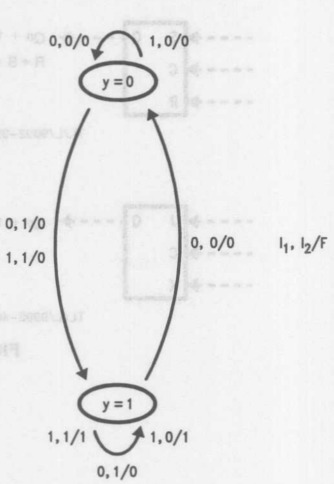
TL/L/9992–42

FIGURE A.6.1. A Typical Sequential Circuit

This system has two primary input variables, having four different combinations of values. There is one primary output variable and one state variable. It uses delay for memory. There are only two possible present states: y = 0 and y = 1. When combined with the four input combinations, these give eight different total present states. The values of the next-state variable, Y, and the primary output variable, F, must be specified for each total present state. The tabular arrangement shown in Table A.6.1 is a common method for presenting this information. This descriptive tool is called a state table.

TABLE A.6.1. State Table

Present State	Next-State Y	Output F
y	I_1I_2 = 00 01 10 11	I_1I_2 = 00 01 10 11
0	0 1 0 1	0 0 0 0
1	0 1 1 1	0 0 1 1



TL/L/9992–43

FIGURE A.6.2. State Diagram

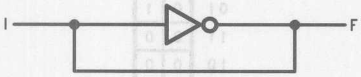
A.6 State Machine Fundamentals (Continued)

A second method for describing the behavior of a sequential system is the use of a state diagram. This method presents a pictorial representation of the present-state/next-state sequences that apply to the sequential device. State changes are marked with directed arrows, with the primary input and output conditions that apply to each state transfer given beside the arrows. The state diagram for the system of *Figure A.6.1* is shown in *Figure A.6.2*. A slash separates the input information from the output information.

State tables and state diagrams are essential tools in the analysis and design of sequential digital systems. The reader should be familiar with these two tools by reading the references listed in the end of this section.

Because a sequential system has feedback from its outputs to its input, certain types of instabilities and uncertainties can occur. When present, these conditions make the operation of circuit difficult or impossible to describe. They may even render the circuit useless, since its behavior may not be predictable or consistent. Several of these types of problems are listed below.

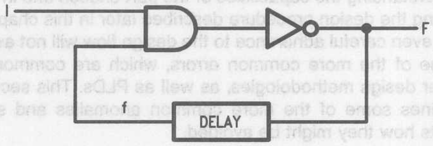
1. The input or output condition of the system may be indeterminant. For example, the circuit in *Figure A.6.3*.



TL/L/9992–44

FIGURE A.6.3. Example of Hazard Circuit

2. The output condition of the system may be unstable, changing even though the external inputs do not change. *Figure A.6.4* illustrates an example.



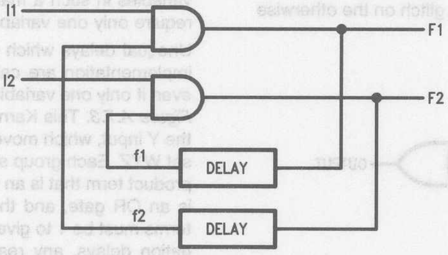
TL/L/9992–45

FIGURE A.6.4. Example of Unstable Circuit

3. The output condition of the system, even though stable, may not be predictable depending upon the primary input conditions. *Figure A.6.5* is an example.

However, these problems mentioned above can be avoided by making certain restrictions in the way sequential systems are designed and used. For instance, the following are some restrictions:

1. Avoiding continuing instabilities (oscillations).
2. Allowing only fundamental-mode operation.
3. Allowing only pulse-mode operation.



TL/L/9992–46

FIGURE A.6.5. Example of Circuit with Unpredictable Output States

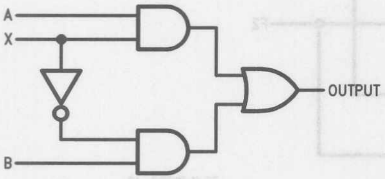
lowing the design procedure described later in this chapter. But even careful adherence to the design flow will not avoid some of the more common errors, which are common in other design methodologies, as well as PLDs. This section outlines some of the more common anomalies and suggests how they might be avoided.

HAZARDS AND GLITCHES

Not all devices have the same propagation delay. A hazard may be caused by configuring a set of gates such that a change in the input signals can cause a spurious output signal or "glitch". In combinational circuits, the hazard will be prevented since the outputs are presumed to be a function of steady-state input signals and are not scanned until all transients have stabilized. However, in sequential circuits, particularly where the outputs of such a combinational circuit are used as inputs to a sequential circuit, glitches may occur.

STATIC AND DYNAMIC HAZARDS

Depending on the initial and final value of the output, there can be two classes of hazards. When these values are the same, extraneous output signals result from a *static* hazard. As an example, the circuit shown in *Figure A.7.1* will exhibit an output glitch due to a static hazard when both inputs A and B are high and the control input is changed from high to low. In a perfect world, the output signal would not change, but the propagation delay of the logic gates (in this case the inverter) will cause a momentary low glitch on the otherwise high output, as shown.



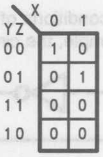
TL/L/9992–1

FIGURE A.7.1. Circuit with Static Hazard

If the initial and final states of the output of a circuit are different, then an extraneous output results from a *dynamic*

FUNCTION AND LOGIC HAZARDS

The causes of hazards are classed as either function or logic. *Function* hazards exist when logic is specified with a change in more than one input variable possible simultaneously. *Figure A.7.2* shows a truth table which illustrates this. The circuit is intended to move from stable state XYZ = 000 to stable state XYZ = 101. If the input variable X and Z do not change absolutely simultaneously, an output glitch due to a function hazard will occur. Assume both X and Z transition from 0 to 1 at about the same time, but not simultaneously. If X changes before Z, a momentary state of 100 will exist, giving a transient output of 0 until Z changes and the final output stabilizes at 1. If Z changes before X, the inputs are momentarily 001, which gives an output 0, which changes to 1 as X changes.



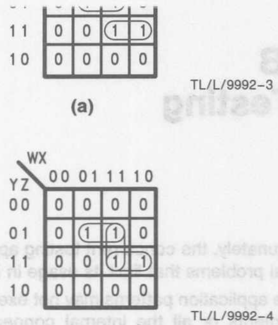
YZ \ X	0	1
00	1	0
01	0	1
11	0	0
10	0	0

TL/L/9992–2

FIGURE A.7.2. Truth Table Illustrating a Function Hazard

Functional glitches can be avoided by assigning the state variables in such a manner that transitions between states require only one variable to change at a time.

Unequal delays which occur because of the detailed logic implementation are called *logic* hazards. These can exist even if only one variable at a time changes, as illustrated by *Figure A.7.3*. This Karnaugh map displays a logic hazard in the Y input, which moves the circuit from the set $X\overline{Y}Z$ to the set WYZ. Each group shown in *Figure A.7.3* represents one product term that is an input to the circuit. In this example, it is an OR gate, and therefore at least one of the product terms must be 1 to give an output of 1. Due to circuit propagation delays, any real-world circuit will move out of the starting sets faster than it moves into the final sets. There is therefore the possibility of a brief interval when neither corresponding product is at 1.



(a)

(b)

FIGURE A.7.3 Karnaugh Map (K-Map) Used to Resolve a Function Hazard

A remedy for this is to ensure that any pair between which a transition may take place are in a single set. In other words, any 1-values which appear next to each other in the K-map must be contained within the same set, as shown in *Figure A.7.3*.

REMEDIES FOR MORE COMPLEX CIRCUITS

Once the number of terms exceeds two or three, K-maps become increasingly difficult to work with. A remedy for this can be found by adding additional terms to the original Boolean equations. From this, it can be determined whether a logic hazard exists by examining the modified equations. If a variable and its complement appear in separate product terms in the same equation and these product terms contain that are not mutually exclusive, a logic hazard exists. The hazard can be eliminated by generating a new product term to overlay each pair of product terms which pose a logic hazard. The new product term is selected from canonical product terms which differ only by the state of the variable causing the hazard.

Hazards can exist irrespective of the design methodology used. In manual design, generation and careful examination of K-maps, particularly multiple inputs for state change, can reveal potential hazards. Computer-aided design tools such as ABEL and CUPL are not completely hazard-free and a similar examination of their results may reveal hazards and require adjustment of minimization level and the addition of redundant terms, as for manual design.

As an example of hazard recognition and correction, consider the circuit shown in *Figure A.7.4*. The Boolean equation describing this is:

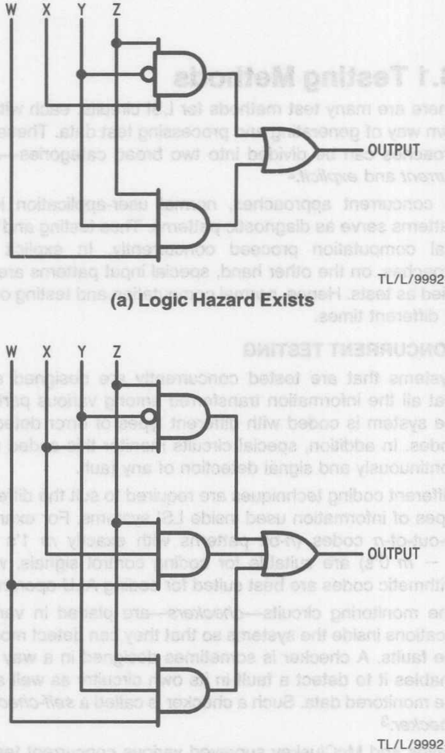
$$X\overline{Y}Z + WYZ$$

Examining the equation reveals a logic hazard because both Y and $\overline{Y}$ appear in separate product terms and inputs W and X are not mutually exclusive. The problem can be eliminated in two steps. Firstly, expand the expression to its canonical form, which gives:

$$WX\overline{Y}Z + \overline{W}X\overline{Y}Z + WXYZ + W\overline{X}YZ$$

$$= X\overline{Y}Z + WXZ + WYZ$$

In this case, the new product term WXZ overlays the original and is illustrated on the K-map of *Figure A.7.3*. Therefore, the addition of an AND gate and an input to the OR gate will result in elimination of the hazard, as shown in *Figure A.7.4*.



(a) Logic Hazard Exists

(b) No Logic Hazard

FIGURE A.7.4. Recognition and Correction of a Logic Hazard

References

Hill & Peterson, "Introduction to Switching Theory and Logical Design".

Kohavi, "Switching and Finite Automata Theory"

Rhyne, "Fundamentals of Digital Systems Designs".

Krieger, "Basic Switching Circuit Theory".

National Semiconductor

Appendix B
Theory of PLD Testing

B.1 Testing Methods

There are many test methods for LSI circuits, each with its own way of generating and processing test data. These approaches can be divided into two broad categories—*concurrent* and *explicit*.[2]

In concurrent approaches, normal user-application input patterns serve as diagnostic patterns. Thus testing and normal computation proceed concurrently. In explicit approaches, on the other hand, special input patterns are applied as tests. Hence, normal computation and testing occur at different times.

CONCURRENT TESTING

Systems that are tested concurrently are designed such that all the information transferred among various parts of the system is coded with different types of error detecting codes. In addition, special circuits monitor this coded data continuously and signal detection of any fault.

Different coding techniques are required to suit the different types of information used inside LSI systems. For example *m*-out-of-*n* codes (*n*-bit patterns with exactly *m* 1's and $n - m$ 0's) are suitable for coding control signals, while arithmetic codes are best suited for coding ALU operands.[3]

The monitoring circuits—*checkers*—are placed in various locations inside the systems so that they can detect most of the faults. A checker is sometimes designed in a way that enables it to detect a fault in its own circuitry as well as in the monitored data. Such a checker is called a *self-checking checker.*[3]

Hayes and McCluskey surveyed various concurrent testing methods that can be used with microprocessor-based LSI systems.[2] Concurrent testing approaches provide the following advantages:

- Explicit testing expenses (e.g., for test equipment, down time, and test pattern generation) are eliminated during the life of the system, since the data patterns used in normal operation serve as test patterns.
- The faults are detected instantaneously during the use of the LSI chip, hence the first faulty data pattern caused by a certain fault is detected. Thus, the user can rely on the correctness of his output results within the degree of fault coverage provided by the error detection code used. In explicit approaches, on the other hand, nothing can be said about the correctness of the results until the chip is explicitly tested.
- Transient faults, which may occur during normal operation, are detected if they cause any faulty data pattern. These faults cannot be detected by any explicit testing method.

Unfortunately, the concurrent testing approach suffers from several problems that limit its usage in LSI testing:

- The application patterns may not exercise all the storage elements or all the internal connection lines. Defects may exist in places that are not exercised, and hence the faults these defects would produce will not be detected. Thus, the assumption that faults are detected as they occur, or at least before any other fault occurs, is no longer valid. Undetected faults will cause fault accumulation. As a result, the fault detection mechanism may fail because most error detection codes have a limited capability for detecting multiple faults.
- Using error detecting codes to code the information signals used in an LSI chip requires additional I/O pins. At least two extra pins are needed as error signal indicators. (A single pin cannot be used, since such a pin stuck at the good value could go undetected). Because of constraints on pin count, however, such requirements cannot be fulfilled.
- Additional hardware circuitry is required to implement the checkers and to increase the width of the data carriers used for storing and transferring the coded information.
- Designing an LSI circuit for concurrent testing is a much more complicated task than designing a similar LSI circuit that will be tested explicitly.
- Concurrent approaches provide no control over critical voltage or timing parameters. Hence, devices cannot be tested under marginal timing and electrical conditions.
- The degree of fault coverage usually provided by concurrent methods is less than that provided by explicit methods.

The above-mentioned problems have limited the use of concurrent testing for most commercially available LSI circuits. However, as digital systems grow more complex and difficult to test, it becomes increasingly attractive to build test procedures into the UUT (unit under test) itself. We will not consider the concurrent approach further in this article. For a survey of work in concurrent testing, see Hayes and McCluskey.[2]

EXPLICIT TESTING

All explicit testing methods separate the testing process from normal operation. In general, an explicit testing process involves three steps:

- **Generating the test patterns.** The goal of this step is to produce those input patterns which will exercise the UUT under different modes of operation while trying to detect any existing fault.

- **Applying the test patterns to the UUT.** There are two ways to accomplish this step. The first is external testing—the use of special test equipment to apply the test patterns externally. The second is internal testing—the application of test patterns internally by forcing the UUT to execute a self-testing procedure.[2] Obviously, the second method can only be used with systems that can execute programs (for example, with microprocessor-based systems). External testing gives better control over the test process and enables testing under different timing and electrical conditions. On the other hand, internal testing is easier to use because it does not need special test equipment or engineering skills.
- **Evaluating the responses obtained from the UUT.** This step is designed with one of two goals in mind. The first is the detection of an erroneous, which indicates the existence of one or more faults *(go/no-go testing)*. The other is the isolation of the fault, if one exists, in an easily replaceable module *(fault location testing)*. Our interest in this article will be go/no-go testing, since fault location testing of LSI circuits sees only limited use.

Many explicit test methods have evolved in the last decade. They can be distinguished by the techniques used to generate the test patterns and to detect and evaluate the faulty responses *(Figure B.1.1)*. In what follows, we concentrate on explicit testing and present in-depth discussions of the methods of test generation and response evaluation employed with explicit testing.

B.2 Test Generation Techniques

The test generation process represents the most important part of any explicit testing method. Its main goal is to generate those test patterns that, when applied to the UUT, sensitize existing faults and propagate a faulty response to an observable output of the UUT. A test sequence is considered good if it can detect a high percentage of the possible UUT faults; it is considered good, in other words, if its degree of *fault coverage* is high.

Rigorous test generation should consist of three main activities:

- Selecting a good descriptive model, at a suitable level, for the system under consideration. Such a model should reflect the exact behavior of the system in all its possible modes of operation.
- Developing a fault model to define the types of faults that will be considered during test generation. In selecting a fault model, the percentage of possible faults covered by the model should be maximized, and the test costs associated with the use of the model should be minimized. The latter can be accomplished by keeping the complexity of the test generation low and the length of the tests short. Clearly these objectives contradict one another—a good fault model is usually found as a result of a trade-off between them. The nature of the fault model is usually influenced by the model used to describe the system.
- Generating tests to detect all the faults in the fault model. This part of test generation is the soul of the whole test process. Designing a test sequence to detect a certain fault in a digital circuit usually involves two problems. First, the fault must be *excited;* i.e., a certain test sequence must be applied that will force a faulty value to appear at the fault site if the fault exists. Second, the test must be *made sensitive* to the fault; i.e., the effect of the fault must propagate through the network to an observable output.

Rigorous test generation rests heavily on both accurate descriptive (system) models and accurate fault models.

Test generation for digital circuits is usually approached either at the gate-level or at the functional level. The classical approach of modeling digital circuits as a group of connected gates and flip-flops has been used extensively. Using this level of description, test designers introduced many types of fault models, such as the classical stuck-at model. They also assumed that such models could describe physical circuit failures in terms of logic. This assumption has sometimes restricted the number of physical failures that can be modeled, but it has also reduced the complexity of test generation since failures at the elementary level do not have to be considered.

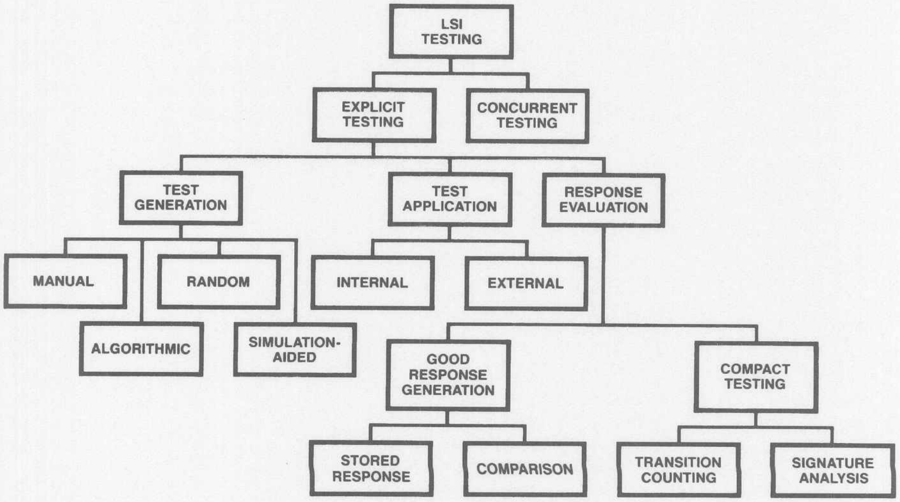


FIGURE B.1.1. LSI Test Technology

NP-COMPLETE PROBLEMS

The theory of NP-completeness is perhaps the most important theoretical development in algorithm research in the past decade.[29] Its results have meaning for all researchers who are developing computer algorithms.

It is an unexplained phenomenon that for many of the problems we know and study, the best algorithms for their solution have computing times which cluster into two groups. The first group consists of problems whose solution is bounded by a polynomial of small degree. Examples include ordered searching, which is $0(\log n)$, polynomial evaluation, which is $0(n)$, and sorting, which is $0(n \log n)$.[30]

The second group contains problems whose best-known algorithms are nonpolynomial. For example, the best algorithms described in Horowitz and Sahni's book[2] for the traveling salesman and the knapsack problems have a complexity of $0(n^2 2^n)$ and $0(2^{n/2})$, respectively. In the quest to develop efficient algorithms, no one has been able to develop a polynomialtime algorithm for any problem in the second group.

The theory of NP-completeness does not provide a method for obtaining polynomial-time algorithms for these problems. But neither does it say that algorithms of this complexity do not exist. What it does show is that many of the problems for which there is no known polynomial-time algorithm are computationally related. In fact, a problem that is NP-complete has the property that it can be solved in polynomial time if all other NP-complete problems can also be solved in polynomial time.

REFERENCES

29. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1978.

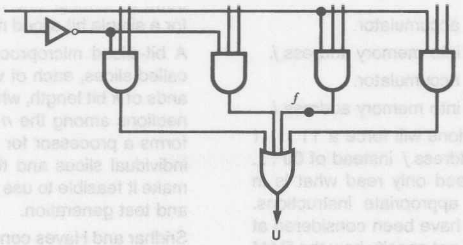
30. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Washington, D.C., 1978.

Many algorithms have been developed for generating tests for a given fault in combinational networks.[1, 4, 5, 6, 7] However, the complexity of these algorithms depends on the topology of the network; it can become very high for some circuits. Ibarra and Sahni have shown that the problem of generating tests to detect single stuck-at faults in a combinational circuit modeled at the gate level is an NP-complete problem.[8] Moreover, if the circuit is sequential, the problem can become even more difficult depending on the deepness of the circuit's sequential logic.

Thus, for LSI circuits having many thousands of gates, the gate level approach to the test generation problem is not very feasible. A new approach, the functional level, is needed.

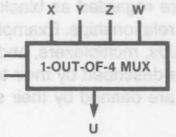
Another important reason for considering faults at the functional level is the constraint imposed on LSI testing by a user environment—the test patterns have to be generated without a knowledge of the implementation details of the chip at the gate level. The only source of information usually available is the typical IC catalog, which details the different modes of operation and describes the general architecture of the circuit. With such information, the test designer finds it easier to define the functional behavior of the circuit and to associate faults with the functions. He can partition the UUT into various modules such as registers, multiplexers, ALUs, ROMs, and RAMs. Each module can be treated as a "black box" performing a specified input/output mapping. These modules can then be tested for *functional failures;* explicit consideration of faults affecting the internal lines is not necessary. The example given below clarifies the idea.

Consider a simple one-out-of-four multiplexers such as the one shown in *Figure B.2.1.* This multiplexer can be modeled at the gate level as shown in *Figure B.2.1(a)*, or at the functional level as shown in *Figure B.2.1.(b)*.



TL/L/9993-2

(a) Gate-Level Description



TL/L/9993-3

C_1	C_0	U
0	0	X
0	1	Y
1	0	Z
1	1	W

(b) Functional-Level Description

FIGURE B.2.1. A One-Out-of-Four Multiplexer

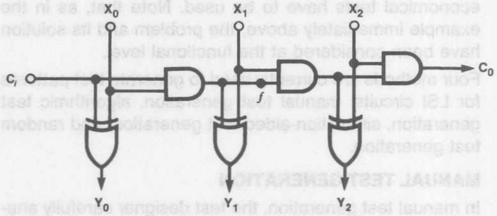
A possible fault model for the gate-level description is the single stuck-at fault model. With this model, the fault list may contain faults such as the line labeled with f is stuck at 0, or the control line "C_0" is stuck at 1.

At the functional level, the multiplexer is considered a black box with a well-defined function. Thus, a fault model for it may specify the following as possible faults: selection of wrong source, selection of no source, or presence of stuck-at faults in the input lines or in the multiplexer output. With this model, the fault list may contain faults such as source "X" is selected instead of source "Y", or line "Z" is stuck at 1.

Ad hoc methods—which determine what faults are the most probable—are sometimes used to generate fault lists. But if no fault model is assumed, then the tests derived must be either exhaustive or a rather ad hoc check of the functionality of the system. Exhaustive tests are impossible for even small systems because of the enormous number of possible states, and superficial tests provide neither good coverage nor even an indication of what faults are covered.

Once the fault list has been defined, the next step is to find the test patterns required to detect the faults in the list. As previously mentioned, each fault first has to be excited so that an error signal will be generated somewhere in the UUT. Then this signal has to be sensitized at one of the observable outputs of the UUT. The three examples below describe how to excite and sensitize different types of faults in the types of modules usually encountered in LSI circuits.

Consider the gate-level description of the three-bit incrementer shown in *Figure B.2.2.*



TL/L/9993-4

FIGURE B.2.2. Gate-Level Description of Three-Bit Incrementer

The incrementer output, $Y_2Y_1Y_0$ is the binary sum of C_i and the three-bit binary number $X_2X_1X_0$, while C_0 is the carry-out bit of the sum. Note that $X_0(Y_0)$ is the least significant bit of the incrementer input (output).

Assume we want to detect the fault "line f is stuck at 0." To excite that fault we will force a 1 to appear on line f so that, if it is stuck at 0, a faulty value will be generated at the fault site. To accomplish this both X_0 and C_i must be set to 1. To sensitize the faulty 0 at f, we have to set X_1 to 1; this will propagate the fault to Y_2 independent of the value of X_2. Note that if we set X_1 to 0, the fault will be masked since the AND gate output will be 0, independent of the value at f. Note also that X_2 was not specified in the above test. However, by setting X_2 to 1, the fault will propagate to both Y_2 and C_0, which makes the response evaluation task easier.

Consider a microprocessor RAM and assume we want to generate a test sequence to detect the fault "accessing word *i* in the RAM results in accessing the word *j* instead." To excite such a fault, we will use the following sequence of instructions (assume a microprocessor with single-operand instructions):

Load the word 00 . . . 0 into the accumulator.

Store the accumulator contents into memory address *j*.

Load the word 11 . . . 1 into the accumulator.

Store the accumulator contents into memory address *i*.

If the fault exists, these instructions will force a 11 . . . 1 word to be stored in memory address *j* instead of 00 . . . 0. To sensitize the fault, we need only read what is in memory address *j*, using the appropriate instructions. Note that the RAM and its fault have been considered at the functional level, since we did not specify how the RAM is implemented.

Consider the program counter (PC) of a microprocessor and assume we want to generate a test sequence that will detect any fault in the incrementing mode of this PC, i.e., any fault that makes the PC unable to be incremented from *x* to *x* + 1 for any address *x*. One way to excite this fault is to force the PC to step through all the possible addresses. This can be easily done by initializing the PC to zero and then executing the no-operation instruction *x* + 1 times. As a result, the PC will contain an address different than *x* + 1. By executing another no-operation instruction, the wrong address can be observed at the address bus and the fault detected. In practice, such an exhaustive test sequence is very expensive, and more economical tests have to be used. Note that, as in the example immediately above, the problem and its solution have been considered at the functional level.

Four methods are currently used to generate test patterns for LSI circuits: manual test generation, algorithmic test generation, simulation-aided test generation, and random test generation.

MANUAL TEST GENERATION

In manual test generation, the test designer carefully analyzes the UUT. This analysis can be done at the gate level, at the functional level or at a combination of the two. The analysis of the different parts of the UUT is intended to determine the specific patterns that will excite and sensitize each fault in the fault list. At one time, the manual approach was widely used for medium- and small-scale digital circuits. Then, the formulation of the D-algorithm and similar algorithms eliminated the need for analyzing each circuit manually and provided an efficient means to generate the required test patterns.[15] However, the arrival of LSI circuits and microprocessors required a shift back toward manual test generation techniques, because most of the algorithmic techniques used with SSI and MSI circuits were not suitable for LSI circuits.

Manual test generation tends to optimize the length of the test patterns and provides a relatively high degree of fault coverage. However, generating tests manually takes a considerable amount of effort and requires persons with special skills. Realizing that test generation has to be done economically, test designers are now moving in the direction of automatic test generation.

One good example of manual test generation is the work done by Sridhar and Hayes,[9] who generated test patterns for a simple bit-sliced microprocessor at the functional level.

A bit-sliced microprocessor is an array of *n* identical ICs called slices, each of which is a simple processor for operands of *k* bit length, where *k* is typically 2 or 4. The interconnections among the *n* slices are such that the entire array forms a processor for *nk*bit operands. The simplicity of the individual slices and the regularity of the interconnections make it feasible to use systematic methods for fault analysis and test generation.

Sridhar and Hayes considered a one-bit processor slice as a simplified model for the commercially available bit-sliced processors such as the Am2901.[10] A slice can be modeled as a collection of modules interconnected in a known way. These modules are regarded as black boxes with well-defined input-output relationships. Examples of these functional modules are ALUs, multiplexers, and registers. Combinational modules are described by their truth tables, while sequential modules are defined by their state tables (or state diagrams).

The following fault categories were considered:

- For combinational modules, all possible faults that induce arbitrary changes in the truth table of the module, but that cannot convert it into a sequential circuit.
- For sequential modules, all possible faults that can cause arbitrary changes in the state table of the module without increasing the number of states.

Only one module was assumed to be faulty at any time.

To test for the faults allowed by the above-mentioned fault model, all possible input patterns must be applied to each combinational module (exhaustive testing), and a checking sequence[11] to each sequential module. In addition, the responses of each module must be propagated to observable output lines. The tests required by the individual modules were easily generated manually—a direct consequence of the small operand size ($k = 1$). And because the slices were identical, the tests for one slice were easily extended to the whole array of slices. In fact, Sridhar and Hayes showed that an arbitrary number of simple interconnected slices could be tested with the same number of tests as that required for a single slice, as long as only one slice was faulty at one time. This property is called *C-testability*. Note that the use of carry-lookahead when connecting slices eliminates C-testability. Also note that slices with operand sizes equal to 2 or more usually are not C-testable.

The idea of modeling a digital system as a collection of interconnected functional modules can be used in modeling any LSI circuit. However, using exhaustive tests and checking sequences to test individual modules is feasible only for toy systems. Hence, the fault model proposed by Sridhar and Hayes, though very powerful, is not directly applicable to LSI testing.

PATH SENSITIZATION AND THE D-ALGORITHM

One of the classical fault detection methods at the gate and flip-flop level is the D-algorithm[1, 5] employing the path sensitization testing technique.[4] The basic principle involved in path sensitization is relatively simple. For an input X_i to detect a fault "line a is stuck at j, $j = 0, 1$," the input X_i must cause the signal a in the normal (fault-free) circuit to take the value $\bar{j}$. This condition is necessary but not sufficient to detect the fault. The error signal must be propagated along some path from its site to an observable output.

To generate a test to detect a stuck-at fault in a combinational circuit, the following path sensitization procedure must be followed:

- Excitation—The inputs must be specified so as to generate the appropriate value (0 for stuck-at 1 and 1 for stuck-at 0) at the site of the fault.
- Error propagation—A path from the fault site to an observable output must be selected, and additional signal values to propagate the fault signal along this path must be specified.
- Error propagation—A path from the fault site to an observable output must be selected, and additional signal values to propagate the fault signal along this path must be specified.
- Line justification—Input values must be specified so as to produce the signals values specified in the step above.

There may be several possible choices for error propagation and line justification. Also, in some cases there may be a choice of ways in which to excite the fault. Some of these choices may lead to an inconsistency, and so the procedure must backtrack and consider the next alternative. If all the alternatives lead to an inconsistency, this implies that the fault cannot be detected.

To facilitate the path sensitization process, we introduce the symbol D to represent a signal which has the value 1 in a normal circuit and 0 in a faulty circuit, and $\overline{D}$ to represent a signal which has the value 0 in a normal circuit and 1 in a faulty circuit. The path sensitization procedure can be formulated in terms of a cubical algebra[1, 5] to enable automatic generation of test. This also facilitates test generation for more complex fault models and for fault propagation through complex logic elements.

We shall define three types of cubes (i.e., line values specified in positional notation):

- For a circuit element E which realizes the combinational function f, the "primitive cubes" offer a typical presentation of the prime implicants of f and $\bar{f}$. These cubes concisely represent the logical behavior of E.
- A "primitive D-cube of a fault" in a logic element E specifies the minimal input conditions that must be applied to E in order to produce an error signal (D or $\overline{D}$) at the output of E.
- The "propagation D-cubes" of a logic element E specify the minimal input conditions to the logic element that are required to propagate an error signal on an input (or inputs) to the output of that element.

To generate a test for a stuck-at fault in a combinational circuit, the D-algorithm must perform the following:

1. Fault excitation—A primitive D-cube of the fault under consideration must be selected. This generates the error signal D or $\overline{D}$ at the site of the fault. (Usually a choice exists in this step. The initial choice is arbitrary, and it may be necessary to backtrack and consider another choice).
2. Implication—In Step 1 some of the gate inputs or outputs may be specified so as to uniquely imply values on other signals in the circuit. The implication procedure is performed both forwards and backwards through the circuit. Implication is performed as follows: Whenever a previously unspecified signal value becomes specified, all the elements associated with this signal are placed on a list B and processed one at a time (and removed). For each element processed, it is determined if new values of 0, 1, D, and $\overline{D}$ are implied, based on the previously specified inputs and outputs. These implied line values are determined by intersecting the test cube (which specifies all the previously determined signal values of the circuit) with the primitive cubes of the element. If any line values are implied, the area specified in the test cube, and the associated gates are placed on the list B. An inconsistency occurs when a value is implied on a line which has been specified previously to a different value. If an inconsistency occurs, the procedure must backtrack to the last point a choice existed, reset all lines to their values at that point, and begin again with the next choice.
3. D-propagation—All the elements in the circuit whose output values are unspecified and whose input has some signal D or $\overline{D}$ are placed on a list caled the D-frontier. In this step, an element from the D-frontier is selected and values are assigned to its unspecified inputs so as to propagate the D or $\overline{D}$ on its inputs to one of its outputs. This is accomplished by intersecting the current test cube describing the circuit signal values with a propagation D-cube of the selected element of the D-frontier, resulting in a new test cube. If such intersection is impossible, a new element in the D-frontier is selected. If intersection fails for all the elements in the D-frontier, the procedure backtracks to the last point at which a choice existed.
4. Implication of D-propagation—Implication is performed for the new test cube derived in Step 3.
5. Steps 3 and 4 are repeated until the faulty signal has been propagated to an output of the circuit.

PATH SENSITIZATION AND THE D-ALGORITHM (Continued)

6. Line justification—Execution of Steps 1 to 5 may result in specifying the output value of an element E but leaving some of the inputs to the element unspecified. The unspecified inputs of such an element are assigned values so as to produce the desired output value. This is done by intersecting the test cube with any primitive cube of the element which has no specified signal values that differ from those of the test cube.
7. Implication of line justification—Implication is performed on the new test cube derived in Step 6.
8. Steps 6 and 7 are repeated until all specified element outputs have been justified. Backtracking may again be required.

REFERENCES

1. M.A Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Washington, D.C., 1976.

4. D.B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinatorial Nets," *IEEE Trans. Electronic Computers*. Vol. EC-15, No. 2, Feb. 1966, pp. 63–73.

5. J.P. Roth, W.G. Bouricius, and P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electronic Computers*, Vol. EC-16, No. 5, Oct. 1967, pp. 567–580.

ALGORITHMIC TEST GENERATION

In algorithmic test generation, the test designer devises a set of algorithms to generate the 1's and 0's needed to test the UUT. Algorithmic test techniques are much more economical than manual techniques. They also provide the test designer with a high level of flexibility. Thus, he can improve the fault coverage of the tests by replacing or modifying parts of the algorithms. Of course, this task is much simpler than modifying the 1's and 0's in a manually generated test sequence.

Techniques that use the gate-level description of the UUT, such as path sensitization[4] and the D-algorithm,[5] can no longer be used in testing complicated LSI circuits. Thus, the problem of generating meaningful sets of tests directly from the functional description of the UUT has become increasingly important. Relatively little work has been done on functional-level testing of LSI chips that are not memory elements.[9,12–17] Functional testing of memory chips is relatively simple because of the regularity of their design and also because their components can be easily controlled and observed from the outside. Various test generation algorithms have been developed to detect different types of faults in memories. [1,18] In the rest of ths section we will concentrate on the general problem of generating tests for irregular LSI chips, i.e., for LSI chips which are not strictly memory chips.

It is highly desirable to find an algorithm that can generate tests for any LSI circuit, or at least most LSI circuits. One good example of work in this area is the technique proposed by Thatte and Abraham for generating tests for microprocessors.[12,13] Another approach, pursued by the authors of this article, is a test generation procedure capable of handling general LSI circuits.[15,16,17]

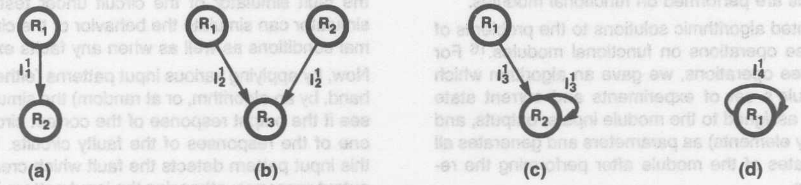
THE THATTE-ABRAHAM TECHNIQUE

Microprocessors constitute a high percentage of today's LSI circuits. Thatte and Abraham[12,13] approached the microprocessor test generation problem at the functional level.

- The test generation procedure they developed was based on:
- A functional description of the microprocessor at the register-transfer level. The model is defined in terms of data flow among storage units during the execution of an instruction. The functional behavior of a microprocessor is thus described by information about its instruction set and the functions performed by each instruction.
- A fault model describing faults in the various functional parts of the UUT (e.g., the data transfer function, the data storage function, the instruction decoding and control function). This fault model describes the faulty behavior of the UUT without knowing its implementation details.

The microprocessor is modeled by a graph. Each register in the microprocessor (including general-purpose registers and accumulator, stack, program counter, address buffer, and processor status word registers) is represented by a node of the graph. Instructions of the microprocessors are classified as being of transfer, data manipulation, or branch type. There exists a directed edge (labeled with an instruction) from one node to another if during an execution of the instruction data flow occurs from the register represented by the first node to that represented by the second. Examples of instruction representation are given in *Figure B.2.3.*

Having described the function or the structure of the UUT, one needs an appropriate fault model in order to derive useful tests. The approach used by Thatte and Abraham is to partition the various functions of a microprocessor into five classes: the register decoding function, the instruction decoding and control function, the data storage function, the data transfer function, and the data manipulation function. Fault models are derived for each of these functions at a higher level and independently of the details of implementation for the microprocessor. The fault model is quite general. Tests are derived allowing any number of faults, but only in one function at a time; this restriction exists solely to cut down the complexity of test generation.



TL/L/9993-5

FIGURE B.2.3. Representations of Microprocessor Instruction—I_1, (a) Transfer Instruction, $R_2 \leftarrow R_1$; (b) Add Instruction, $R_3 \leftarrow R_1 + R_2$; (c) I_3, OR Instruction, $R_2 \leftarrow R_1$ OR R_2; (d) I_4 Rotate Left Instruction

The fault model for the register decoding function allows any possible set of registers to be accessed instead of a particular register. (If the set is null then no register is accessed.) This fault model is thus very general and independent of the actual realization of the decoding mechanism.

For the instruction decoding and control function, the faulty behavior of the microprocessor is specified as follows. When instruction I_j, is executed any one of the following can happen:

- Instead of instruction I_j, some other instruction I_k is executed. This fault is denoted by $F(I_j/I_k)$.
- In addition to instruction I_j, some other instruction I_k is activated. This fault is denoted by $F(I_j/I_j + I_k)$.
- No instruction is executed. This fault is denoted by $F(I_j/\theta)$.

Under this specification, any number of instructions can be faulty.

In the fault model for the data storage function, any cell in any data storage module is allowed to be stuck at 0 or 1. This can occur in any number of cells.

The fault model for the data transfer function includes the following types of faults:

- A line in a path used in the execution of an instruction is stuck at 0 or 1.
- Two lines of a path used in the instruction are coupled, i.e., they fail to carry different logic values.

Note that the second fault type cannot be modeled by single stuck-at faults. The transfer paths in this fault model are logical paths and thus will account for any failure in the actual physical paths.

Since there is a variety of designs for the ALU and other functional units such as increment or shift logic, no specific fault model is used for the data manipulation function. It is assumed that complete test sets can be derived for the functional units for a given fault model.

By carefully analyzing the logical behavior of the microprocessor according to the fault models presented above, Thatte and Abraham formulated a set of algorithms to generate the necessary test patterns. These algorithms step the microprocessor through a precisely defined set of instructions and addresses. Each algorithm was designed for detecting a particular class of faults, and theorems were proved which showed exactly the kind of faults detected by each algorithm. These algorithms employ the excitation and sensitization concepts previously described.

To gain insight into the problems involved in using the algorithms, Thatte investigated the testing of an eight-bit microprocessor from Hewlett-Packard.[12] He generated the test patterns for the microprocessor by hand, using the algorithms. He found that 96 percent of the single stuck-at faults that could affect the microprocessor were detected by the test sequence he generated. This figure indicates the validity of the technique.

THE ABADIR-REGHBATI TECHNIQUE

Here we will briefly describe a test generation technique we developed for LSI circuits.[15,16] We assume that the tests would be generated in a user environment in which the gate- and flip-flop-level details of the chip were not known.

We developed a module-level model for LSI circuits. This model bypasses the gate and flip-flop levels and directly describes blocks of logic (modules) according to their functions. Any LSI circuit can be modeled as a network of interconnected modules such as counters, registers, ALUs, ROMs, RAMs, multiplexers and decoders.

Each module in an LSI circuit was modeled as a black box having a number of functions defined by a set of *binary decision diagrams* (see box).[19] This type of diagram, a functional description tool introduced by Akers in 1978, is a concise means for completely defining the logical operation of one or more digital functions in an implementation-free form. The information usually found in an IC catalog is sufficient to derive the set of binary decision diagrams describing the functions performed by the different modules in a device. These diagrams—like truth tables and state tables—are amenable to extensive logical analysis. However, unlike truth tables and state tables—are amenable to extensive logical analysis. However, unlike truth tables and state tables, they do not have the unpleasant property of growing exponentially with the number of variables involved. Moreover, the diagrams can be stored and processed easily in a digital computer. An important feature of these diagrams is that they state exactly how the module will behave in every one of its operation modes. Such information can be extracted from the module's diagrams in the form of a set of *experiments*.[15,20] Each of these experiments describes the behavior of the module in one of its modes of operation. The structure of these experiments makes them suitable for use in automatic test generation.

We also developed a functional-level fault model describing faulty behavior in the different modules of an LSI chip. This model is quite independent of the details of implementation and covers functional faults that alter the behavior of a module during one of its modes of operation. It also covers stuck-at faults affecting any input or output pin or any interconnection line in the chip.

Using the above-mentioned models, we proposed a functional test generation procedure based on path sensitization and D-algorithm.[15] The procedure takes the module-level model of the LSI chip and the functional description of its modules as parameters and generates tests to detect faults in the fault model. The *fault collapsing technique*[1] was used to reduce the length of the test sequence. As in the D-algorithm, the procedure employs three basic operations, namely implication, D-propagation, and line justification. However, these operations are performed on functional modules.

We also presented algorithmic solutions to the problems of performing these operations on functional modules.[16] For each of the three operations, we gave an algorithm which takes the module's set of experiments and current state (i.e., the values assigned to the module inputs, outputs, and internal memory elements) as parameters and generates all the possible states of the module after performing the required operation.

We have also reported our efforts to develop test sequences based on our test generation procedure for typical LSI circuits.[17] More specifically, we considered a one-bit microprocessor slice C that has all the basic features of the four-bit Am2901 microprocessor slice.[10] The circuit C was modeled as a network of eight functional modules: an ALU, a latch register, an addressable register, and five multiplexers. The functions of the individual modules were described in terms of binary decision diagrams or equivalent sets of experiments. Test capable of detecting various faults covered by the fault model were then generated for the circuit C. We showed that if the fault collapsing technique is used, a significant reduction in the length of the final test sequence results.

The test generation effort was quite straightforward, indicating that the technique can be automated without much difficulty. Our study also shows that for a simplified version of the circuit C the length of the test sequence generated by our technique is very close to the length of the test sequence manually generated by Sridhar and Hayes[9] for the same circuit. We also described techniques for modeling some of the features of the Am2909 four-bit microprogram sequencer[10] that are not covered by the circuit C.

The results of our case study were quite promising and showed that our technique is a viable and effective one for generating tests for LSI circuits.

SIMULATION-AIDED TEST GENERATION

Logic simulation techniques have been used widely in the evaluation and verification of new digital circuits. However; an important application of logic simulation is to interpret the behavior of a circuit under a certain fault or faults. This is known as *fault simulation*. To clarify how this technique can be used to generate tests for LSI systems, we will first describe its use with SSI/MSI-type circuits.

To generate a fault simulator for an SSI/MSI circuit, the following information is needed.[1]

- the gate-level description of the circuit, written in a special language;
- the initial conditions of the memory elements; and
- a list of the faults to be simulated, including classical types of faults such as stuck-at faults and adjacent pin shorts.

The above is fed to a simulation package which generates the fault simulator of the circuit under test. The resulting simulator can simulate the behavior of the circuit under normal conditions as well as when any faults exist.

Now, by applying various input patterns (either generated by hand, by an algorithm, or at random) the simulator checks to see if the output response of the correct circuit differs from one of the responses of the faulty circuits. If it does, then this input pattern detects the fault which created the wrong output response; otherwise the input pattern is useless. If an input pattern is found to detect a certain fault, this fault is deleted from the fault list and the process continues until either the input patterns or the faults are finished. At the end, the faults remaining in the fault list are those which cannot be detected by the input patterns. This direclty measures the degree of fault coverage of the input patterns used.

Two examples of this type of logic simulator are LAMP—the Logic Analyzer for Maintenance Planning developed at Bell Laboratories,[21] and the Testaid III fault simulator developed at the Hewlett-Packard Company.[12] Both work primarily at the gate level and simulate stuck-at faults only. One of the main applications of such fault simulators is to determine the degree of fault coverage provided by a test sequence generated by any other test generation technique.

There are two key requirements that affect the success of any fault simulator:

- the existence of a software model for each primitive element of the circuit, and
- the existence of a good fault model for the UUT which can be used to generate a fault list covering most of the actual physical faults.

These two requirements have been met for SSI/MSI circuits, but they pose serious problems for LSI circuits. If it can be done at all, modeling LSI circuits at the gate level requires great effort. One part of the problem is the lack of detailed information about the internal structure of most LSI chips. The other is the time and memory required to simulate and LSI circuit containing thousands of gates. Another severe problem facing almost all LSI test generation techniques is the lack of good fault models at a level higher than the gate level.

The Abadir-Reghbati description model proposed in the previous section permits the test designer to bypass the gate-level description and, using binary decision diagrams, to define blocks of logic according to their functions. Thus, the simulation of complex LSI circuits can take place at a higher level, and this eliminates the large time and memory requirements. Furthermore, the Abadir-Reghbati fault model is quite efficient and is suitable for simulation purposes. In fact, the implication operation[16] employed by the test generation procedure represents the main building block of any fault simulator. It must be noted that fault simulation techniques are very useful in optimizing the length of the test sequence generated by any test generation technique.

BINARY DECISION DIAGRAMS

Binary decision diagrams are a means of defining the logical operation of digital functions.[19] They tell the user how to determine the output value of a digital function by examining the values of the inputs. Each node in these diagrams is associated with a binary variable, and there are two branches coming out from each node. The right branch is the "1" branch, while the left branch is the "0" branch. Depending on the value of the node variable, one of the two branches will be selected when the diagram is processed.

To see how binary decision diagrams can be used, consider the half-adder shown in *Figure B.2.4(a)*. Assume we are interested in defining a procedure to determine the value of C, given the binary values of X and Y. We can do this by looking at the value of X. If X = 0, then C = 0, and we are finished. If X = 1, we look at Y. If Y = 0, then C = 0, else C = 1, and in either case we are finished. *Figure B.2.4(b)* shows a simple diagram of this procedure. By entering the diagram at the node indicated by the arrow labeled with C and then proceeding through the diagram following the appropriate branches until a 0 or 1 value is reached, we can determine the value C. *Figure B.2.4(c)* shows the diagram representing the function S of the half-adder.

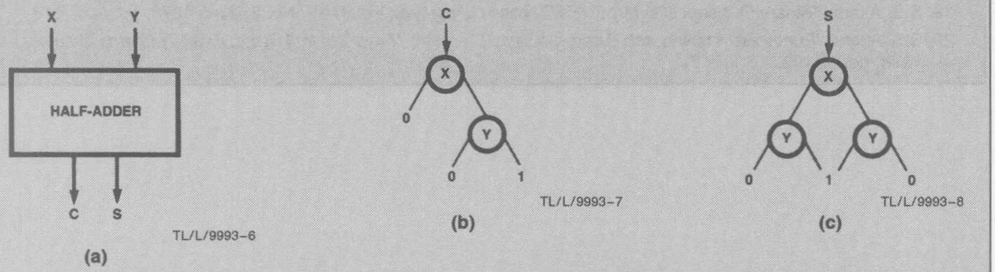


FIGURE B.2.4. (a) Half-Adder; (b) Binary Decision Diagram for C = X • Y; (c) Binary Decision Diagram for S = X ⊕ Y

To simplify the diagrams, any diagram node which has two branches as exit branches can be replaced by the variable itself or its complement. These variables are called exit variables. *Figure B.2.5* shows how this convention is used to simplify the diagrams describing the half-adder.

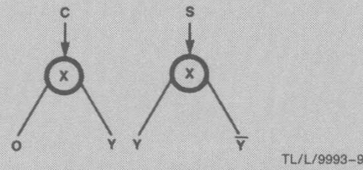


FIGURE B.2.5 Simplified Binary Decision Diagrams for the Half-Adder

In the previous discussion, we have considered only simple diagrams in which the variables within the nodes are primary input variables. However, we can expand the scope of these diagrams by using auxiliary variables as the node variables. These auxiliary variables are defined by their diagrams. Thus, when user encounters such a node variable, say *g*, while tracing a path, he must first process the diagram defining *g* to determine the value of *g*, and then return to the original node and take the appropriate branch. This process is similar to the use of subroutines in high-level programming languages.

For example, consider the full-adder defined by:

$$C_{j+1} = E_j\,C_j + \overline{E}_j\,A_j$$

$$S_j = E_j + C_j,$$

where $E_j = A_j + B_j$,. *Figure B.2.6* shows the diagrams for these three equations. If the user wants to know the value of C_{j+1} when the values of the three primary inputs A_j, B_j, and C are all 1's, he enters the C_{j+1} diagram, where he encounters

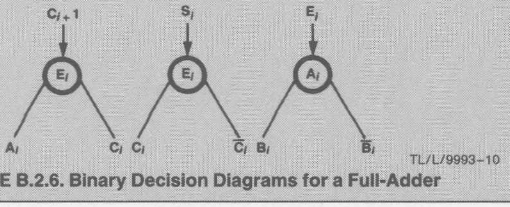


FIGURE B.2.6. Binary Decision Diagrams for a Full-Adder

Since node variables can refer to other auxiliary functions, we can simply describe complex modules by breaking their functions into small subfunctions. Thus, the system diagram will consist of small diagrams connected in a hierarchical structure. Each of these diagrams describes either a module output or an auxiliary variable.

Akers[19] described two procedures to generate the binary decision diagram of a combinational function *f*. The first one uses the truth table description of *f*, while the other uses the boolean expression of *f*. A similar procedure can be derived to generate the binary decision diagram for any sequential function defined by a state table.

Binary decision diagrams can be easily stored and processed by a computer through the use of binary tree structures. Each node can be completely defined by an ordered triple: the node variable and two pointers to the two nodes to which its 0 and 1 branches are directed. Binary decision diagrams can be used in functional testing.[20]

REFERENCES

19. S.B. Akers, "Binary Decision Diagram," *IEEE Trans Computers*, Vol. C-27, No. 6, June 1978, pp. 509–516.

20. S.B. Akers, "Functional Testing with Binary Decision Diagram," *Proc. 8th Int'l Symp. Fault-Tolerant Computing*, June 1978, pp. 82–92.

RANDOM TEST GENERATION

This method can be considered the simplest method for testing a device. A random number generator is used to simultaneously apply random input patterns both to the UUT and to a copy of it known to be fault-free. (This copy is called the *golden unit*.) The results obtained from the two units are compared, and if they do not match, a fault in the UUT is detected. This response evaluation technique is known as comparison testing; we will discuss it later. It is important to note that every time the UUT is tested, a new random test sequence is used.

The important question is how effective the random test is, or, in other words, what fault coverage a random test of given length provides. This question can be answered by employing a fault simulator to simulate the effect of random test patterns of various lengths. The results of such experiments on SSI and MSI circuits show that random test generation is most suitable for circuits without deep sequential logic.[1,22,23] However, by combining random patterns with manually generated ones, test designers can obtain very good results.

The increased sequentiality of LSI circuits reduces the applicability of random testing. Again, combining manually generated test patterns with random ones improves the degree of fault coverage. However, two factors restrict the use of the random test generation technique:

- The dependency on the golden unit, which is assumed to be fault-free, weakens the level of confidence in the results.
- There is no accurate measure of how effective the test is, since all the data gathered about random tests are statistical data. Thus, the amount of fault coverage provided by a particular random test process is unpredictable.

B.3 Response Evaluation Techniques

Different methods have been used to evaluate UUT responses to test patterns. We restrict our discussion to the case where the final goal is only to detect faults or, equivalently, to detect any wrong output response. There are two ways of achieving this goal—using a good response generator or using a compact testing technique.

GOOD RESPONSE GENERATION

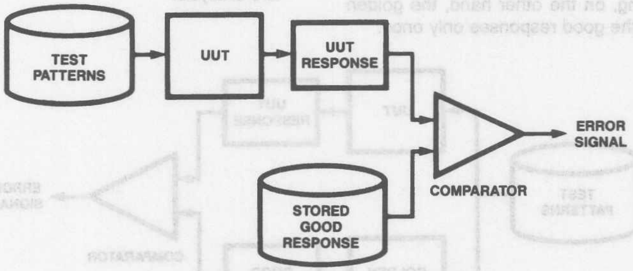
This technique implements an ideal strategy: comparing UUT responses with good response patterns to detect any faulty response. Clearly, the key problems are how to obtain a good response and at what stage in the testing process that response will be generated. In current test systems, two approaches to solving these problems are taken—*stored response testing and comparison testing.*

STORED RESPONSE TESTING

In stored response testing, a one-shot operation generates the good response patterns at the end of the test generation stage. These patterns are stored in an auxiliary memory (usually a ROM). A flow diagram of the stored response testing technique is shown in *Figure B.3.1.*

Different methods can be used to obtain good responses of a circuit to a particular test sequence. One way is to do it manually by analyzing the UUT and the test patterns. This method is the most suitable if the test patterns were generated manually in the first place.

The method most widely used to obtain good responses from the UUT is to apply the test patterns either to a known good copy of the UUT—the golden unit—or to a software-simulated version of the UUT. Of course, if fault simulation techniques were used to generate the test patterns, the UUT's good responses can be obtained very easily as a partial product from the simulator.



TL/L/9993–11

FIGURE B.3.1. Stored Response Testing

The use of a known good device depends on the availability of such a device. Hence, different techniques must be used for the user who wants to test his LSI system and for the designer who wants to test his prototype design. However, golden units are usually available once the device goes into production. Moreover, confidence in the correctness of the responses can be increased by using three or five good devices together to generate the good responses.

The major advantage of the stored response technique is that the good responses are generated only once for each test sequence, thus reducing the cost of the response evaluation step. However, the stored response technique suffers from various disadvantages:

- Any change in the test sequence requires the whole process to be repeated.
- A very large memory is usually needed to store all the good responses to a reasonable test sequence, because both the length and the width of the responses are relatively large. As a result, the cost of testing equipment increases.
- The speed with which the test patterns can be applied to the UUT is limited by the access time of the memory used to store the good responses.

COMPARISON TESTING

Another way to evaluate the responses of the UUT during the testing process is to apply the test patterns simultaneously to both the UUT and a golden unit and to compare their responses to detect any faulty response. The flow diagram of the comparison testing technique is shown in *Figure B.3.2.* The use of comparison testing makes possible the testing of the UUT at different speeds under different electrical parameters, given that these parameters are within the operating limits of the golden unit, which is assumed to be ideal.

Note that in comparison testing the golden unit is used to generate the good responses every time the UUT is tested. In stored response testing, on the other hand, the golden unit is used to generate the good responses only once. The disadvantages of depending on a golden unit are more serious here, however, since every explicit testing process requires one golden unit. This means that every tester must contain a golden copy of each LSI circuit tested by that tester.

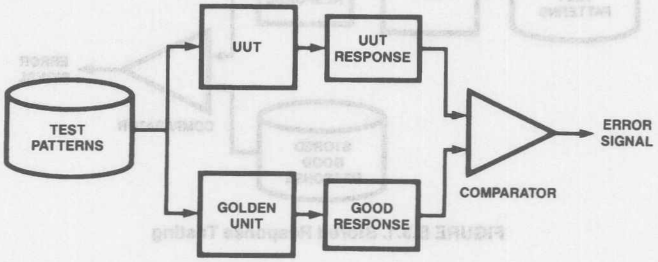
One of the major advantages of comparison testing is that nothing has to be changed in the response evaluation stage if the test sequence is altered This makes comparison testing highly desirable if test patterns are generated randomly.

COMPACT TESTING

The major drawback of good response generation techniques in general, and stored response testing in particular, is the huge amount of response data that must be analyzed and stored. Compact testing methods attempt to solve this by compressing the response data R into a more compact from $f(R)$ from which most of the fault information in R can be derived. Thus, because only the compact form of the good responses has to be stored, the need for large memory or expensive golden units is eliminated. An important property of the compression function f is that it can be implemented with simple circuitry. Thus, compact testing does not require much test equipment and is especially suited for field maintenance work. A general diagram of the compact testing technique is shown in *Figure B.3.3.*

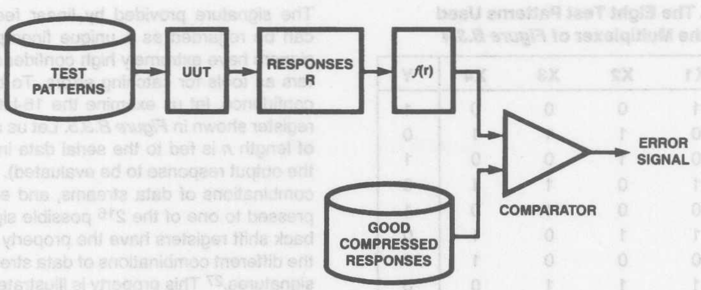
Several choices for the function f exist, such as "the number of 1's in the sequence," "the number of 0 to 1 and 1 to 0 transitions in the sequence" *(transition counting)*,[24] or "the signature of the sequence" *(signature analysis)*.[25] For each compression function f, there is a slight probability that a response R1 different from the fault-free response R0 will be compressed to a form equal to $f(R0)$, i.e., $f(R1) = f(R0)$. Thus, the fault causing the UUT to produce R1 instead of R0 will not be detected, even though it is covered by the test patterns.

The two compression functions that are the most widely accepted commercially are transition counting and signature analysis.



TL/L/9993–12

FIGURE B.3.2. Comparison Testing



TL/L/9993-13

FIGURE B.3.3. Compact Testing

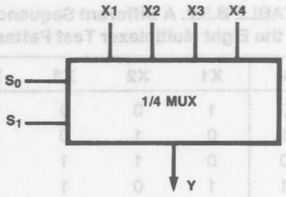
TRANSITION COUNTING

In transition counting, the number of logical transitions (0 to 1 and vice versa) is computed at each output pin by simply running each output of the UUT into a special counter. Thus, the number of counters needed is equal to the number of output pins observed. For every *m*-bit output data stream (at one pin), an *n*-bit counter is required, where $n = [\log_2 m]$. As in stored response testing, the transition counts of the good responses are obtained by applying the test sequence to a golden copy of the UUT and counting the number of transitions at each output pin. This latter information is used as a reference in any explicit testing process.

In the testing of an LSI circuit by means of transition counting, the input patterns can be applied to the UUT at a very high rate, since the response evaluation circuitry is very fast. Also, the size of the memory needed to store the transition counts of the good responses can be very small. For example, a transition counting test using 16 million patterns at a rate of 1 MHz will take 16 seconds, and the compressed stored response will occupy only *K* 24-bit words, where *K* is the number of output pins. This can be contrasted with the 16 million *K*-bit words of storage space needed if regular stored response testing is used.

The test patterns used in a transition counting test system must be designed such that their output responses maximize the fault coverage of the test.[24] The example below shows how this can be done.

Consider the one-out-of-four multiplexer shown in *Figure B.3.4.* To check for multiple stuck-at faults in the multiplexer input lines, eight test patterns are required, as shown in Table B.3.1. The sequence of applying these eight patterns to the multiplexer is not important if we want to evaluate the output responses one by one. However, this sequence will greatly affect the degree of fault coverage if transition counting is used. To illustrate this fact, consider the eight single stuck-at faults in the four input lines X1, X2, X3 and X4 (i.e, X1 stuck-at 0, X1 stuck-at 1, X2 stuck-at 0, and so on). Each of these faults will be detected by only one pattern among the eight test patterns. For example, the fault “X1 stuck-at 0” will be detected by applying the first test pattern in Table B.3.1, but the other seven test patterns will not detect this fault. Now, suppose we want to use transition counting to evaluate the output responses of the multiplexer. Applying the eight test patterns in the sequence shown in Table B.3.1 (from top to bottom) will produce the output response 10101010 (from left to right), with a transition count of seven. Any possible combination of the eight faults described above will change the transition count to a number different from seven, and the fault will be detected. (Note that no more than four of the eight faults can occur at any one time.) Thus, the test sequence shown in Table B.3.1 will detect all single and multiple stuck-at faults in the four input lines of the multiplexers.



TL/L/9993-14

S_0	S_1	Y
0	0	X1
0	1	X2
1	0	X3
1	1	X4

FIGURE B.3.4. One-Out-of-Four Multiplexer

Now, if we change the sequence of the test patterns to the one shown in Table B.3.2., the fault coverage of the test will decrease considerable. The output responses of the sequence of Table B.3.2 will be 11001100, with a transition count of three. As a result, six of the eight single stuck-at faults will not be detected, because the transition count of the six faulty responses will remain three. For example, the fault “X1 stuck-at 1” will change the output response to 11101100, which has a transition count of three. Hence, this fault will not be detected. Moreover, most of the multiple combinations of the eight faults will not change the transition count of the output, and hence they will not be detected either.

It is clear from the above example that the order of applying the test patterns to the UUT greatly affects the fault coverage of the test. When testing combinational circuits, the test designer is completely free to choose the order of test patterns. However, he cannot do the same with test patterns for sequential circuits. More seriously, because he is dealing with LSI circuits that probably have multiple output lines, he will find that a particular test sequence may give good results at some outputs and bad results at others. One way to solve these contradictions is to use simulation techniques to find the optimal test sequence. However, because of the limitations discussed here, transition counting cannot be recognized as a powerful compact LSI testing method.

0	0	0	1	1	1	0
0	1	0	1	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	0	0	1	1
1	1	1	1	1	0	0

TABLE B.3.2. A Different Sequence of the Eight Multiplexer Test Patterns

S_0	S_1	X1	X2	X3	X4	Y
0	0	1	0	0	0	1
0	1	0	1	0	0	1
0	0	0	1	1	1	0
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	1	0	0	0	1	1
1	0	1	1	0	1	0
1	1	1	1	1	0	0

SIGNATURE ANALYSIS

In 1977 Hewlett-Packard Corporation introduced a new compact testing technique called signature analysis, intended for testing LSI systems.[25–28] In this method, each output response is passed through a 16-bit linear feedback shift register whose contents f(R), after the test patterns have been applied, are called the test *signature. Figure B.3.5* shows an example of a linear feedback shift register used in signature analysis.

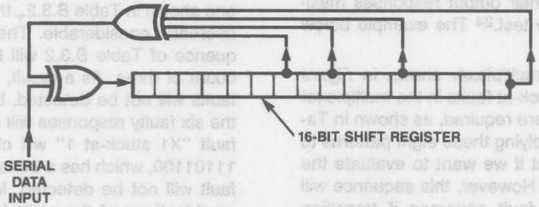
register shown in *Figure B.3.5.* Let us assume a data stream of length n is fed to the serial data input line (representing the output response to be evaluated). There are 2^n possible combinations of data streams, and each one will be compressed to one of the 2^{16} possible signatures. Linear feedback shift registers have the property of equally distributing the different combinations of data streams over the different signatures.[27] This property is illustrated by the following numerical examples.

- Assume $n = 16$. Then each data stream will be mapped to a distinctive signature (one-to-one mapping).
- Assume $n = 17$. Then exactly two data streams will be mapped to the same signature. Thus, for a particular data stream (the UUT good output response), there is only one other data stream (a faulty output response) that will have the same signature; i.e., only one fault response out of $2^{17} - 1$ possible faults will not be detected.
- Assume $n = 18$. Then four different data streams will be mapped to the same signature. Hence, only three faults out of $2^{18} - 1$ possible faults will not be detected.

We can generalize the results obtained above. For any response data stream of length $n > 16$, the probability of missing a faulty response when using a 16-bit signature analyzer is [27]

$$\frac{2^{n-16} - 1}{2^n - 1} \approx 2^{-16}, \text{ for } n >> 16.$$

Hence, the possibility of missing an error in the bit stream is very small (on the order of 0.002 percent). Note also that a great percentage of the faults will affect more than one output pin—hence the probability of not detecting these kind of faults is even lower.



TL/L/9993-15

FIGURE B.3.5. The 16-Bit Linear Feedback Shift Register Used in Signature Analysis

Signature analysis provides a much higher level of confidence for detecting faulty output responses than that provided by transition counting. But, like transition counting, it requires only very simple hardware circuitry and a small amount of memory for storing the good signatures. As a result, the signatures of the output responses can be calculated even when the UUT is tested at its maximum speed. Unlike transition counting, the degree of fault coverage provided by signature analysis is not sensitive to the order of the test patterns. Thus, it is clear that signature analysis is the most attractive solution to the response evaluation problem.

The rapid growth of the complexity and performance of digital circuits presents a testing problem of increasing severity. Although many testing methods have worked well for SSI and MSI circuits, most of them are rapidly becoming obsolete. New techniques are required to cope with the vastly more complicated LSI circuits.

In general, testing techniques fall into the concurrent and explicit categories. In this article, we gave special attention to explicit testing techniques, especially those approaching the problem at the functional level. The explicit testing process can be partitioned into three steps: generating the test, applying the test to the UUT, and evaluating the UUT's responses. The various testing techniques are distinguished by the methods they used to perform these three steps. Each of these techniques has certain strengths and weaknesses.

We have tried to emphasize the range of testing techniques available, and to highlight some of the milestones in the evolution of LSI testing. The details of an individual test method can be found in the source we have cited.

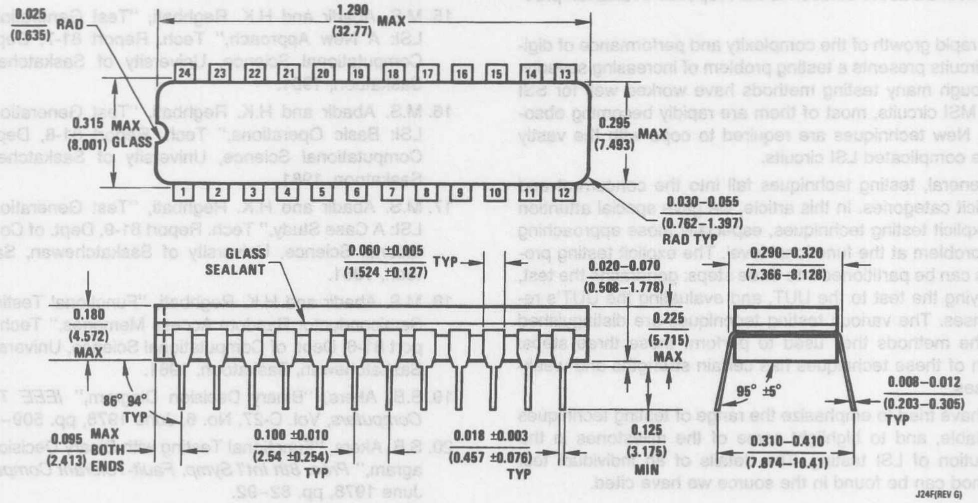
References

1. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Washington, D.C., 1976.
2. J.P. Hayes and E.J. McCluskey, "Testing Considerations in Microprocessor-Based Design", *Computer*, Vol. 13, No.3, March 1980, pp. 17–26.
3. J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, American Elsevier, New York, 1978.
4. D.B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinatorial Nets," *IEEE Trans. Electronic Computers*, Vol. EC-15, No. 2, Feb. 1966, pp. 63–73.
5. J.P. Roth, W.G. Bouricius, and P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electronica Computers*, Vol. EC-16, No. 5, October 1967, pp. 567–580.
6. S.B. Akers, "Test Generation Techniques," *Computer*, Vol. 13, No. 3, March 1980, pp. 9–15.
7. E.I. Muehldorf and A.D. Savkar, "LSI Logic Testing—An Overview," *IEEE Trans. Computers*, Vol. C-30, No. 1, January 1981, pp. 1–17.
8. O.H. Ibarra and S.K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. Computers*, Vol. C-24, No. 3, March 1975, pp. 242–249.
9. T. Sridhar and J.P. Hayes, "Testing Bit-Sliced Microprocessors," *Proc. 9th Int'l Symp. Fault-Tolerant Computing*, 1979, pp. 211–218.
10. *The Am2900 Family Data Book*, Advanced Micro Devices, Inc., 1979.
11. Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1970.
12. S.M. Thatte, "Test Generation for Microprocessors," PhD Thesis, University of Illinois, Urbana, 1979.
13. S.M. Thatte and J.A. Abraham, "Test Generation For Microprocessors," *IEEE Trans. Computers*, Vol. C-29, No. 6, June 1980, pp. 429–441.
14. M.A. Breuer and A.D. Friedman, "Functional Level Primitives in Test Generation," *IEEE Trans. Computers*, Vol. C-29, No. 3, March 1980, pp. 223–235.
15. M.S. Abadir and H.K. Reghbati, "Test Generation for LSI: A New Approach," Tech. Report 81-7, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.
16. M.S. Abadir and H.K. Reghbati, "Test Generation for LSI: Basic Operations," Tech. Report 81-8, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.
17. M.S. Abadir and H.K. Reghbati, "Test Generation for LSI: A Case Study," Tech. Report 81-9, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.
18. M.S. Abadir and H.K. Reghbati, "Functional Testing of Semiconductor Random Access Memories," Tech. Report 81-6, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.
19. S.B. Akers, "Binary Decision Diagram," *IEEE Trans. Computers*, Vol. C-27, No. 6, June 1978, pp. 509–516.
20. S.B. Akers, "Functional Testing with Binary Decision Diagram," *Proc. 8th Int'l Symp. Fault-Tolerant Computing*, June 1978, pp. 82–92.
21. B.A Zimmer, "Test Techniques for Circuit Boards Containing Large Memories and Microprocessors," *Proc. 1976 Semiconductor Test Symp.*, pp. 16–21.
22. P. Agrawal and V.D. Agrawal, "On Improving the Efficiency of Monte Carlo Test Generation," *Proc. 5th Int'l Symp. Fault-Tolerant Computing*, June 1975, pp. 205–209.
23. D. Bastin, E. Girard, J.C. Rault, and R. Tulloue, "Probabilistic Test Generation Methods," *Proc. 3rd Int'l Symp. Fault-Tolerant Computing*, June 1973, p. 171.
24. J.P. Hayes, "Transition Count Testing of Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. C-25, No. 6, June 1976, pp. 613–620.
25. "Signature Analysis," *Hewlett Packard J.*, Vol. 28, No. 9, May 1977.
26. R. David, "Feedback Shift Register Testing," *Proc. 8th Int'l Symp. Fault-Tolerant Computing*, June 1978.
27. H.J. Nadig, "Testing a Microprocessor Product Using Signature Analysis," *Proc. 1978 Semiconductor Test Symp.*, pp. 159–169.
28. J.B. Peatman, *Digital Hardware Design*, McGraw-Hill, New York, 1980.
29. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1978.
30. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Washington D.C., 1978.

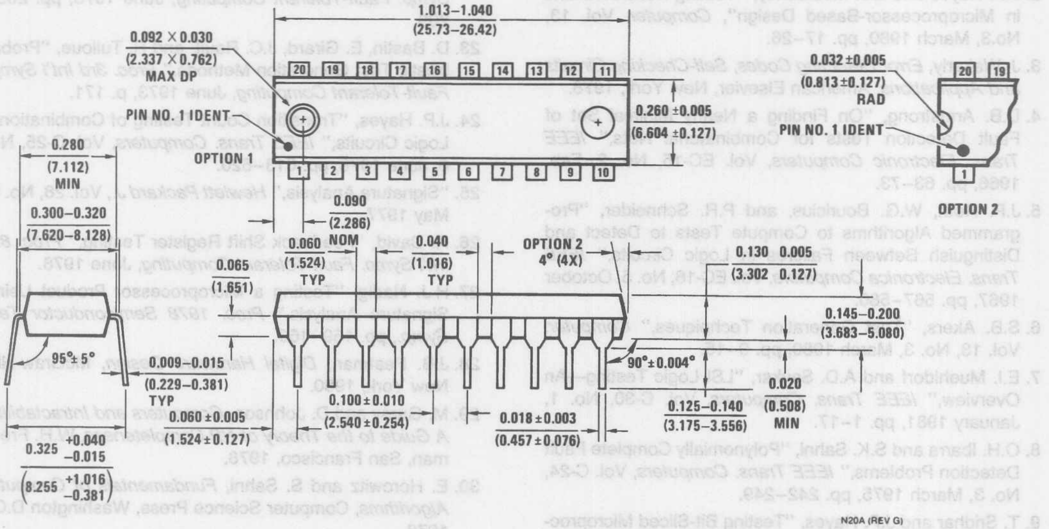
National Semiconductor

All dimensions are in inches (millimeters)

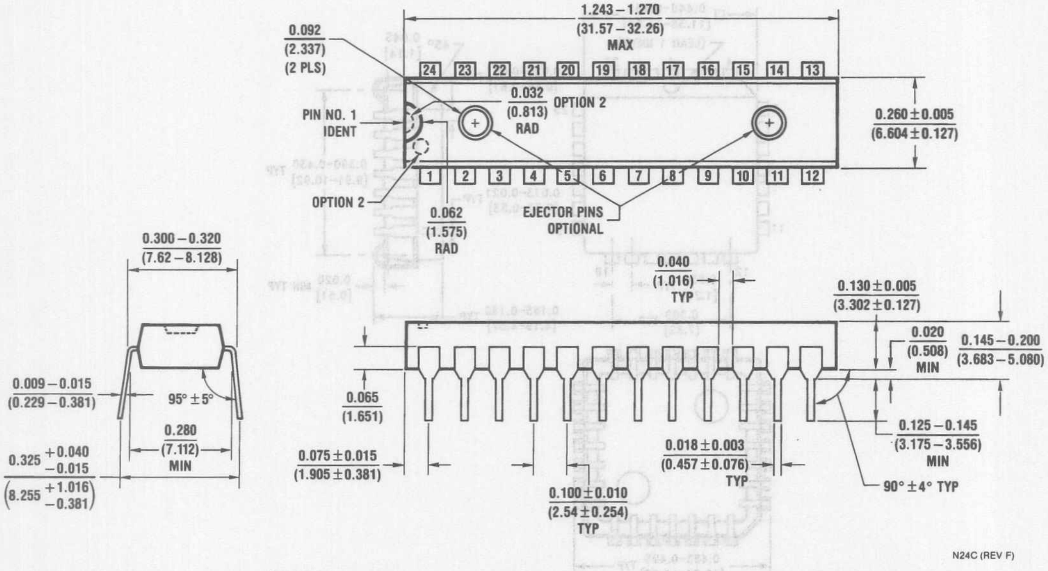
24 Lead Ceramic Dual-In-Line Package (J)
NS Package Number J24F



20 Lead Molded Dual-In-Line Package (N)
NS Package Number N20A

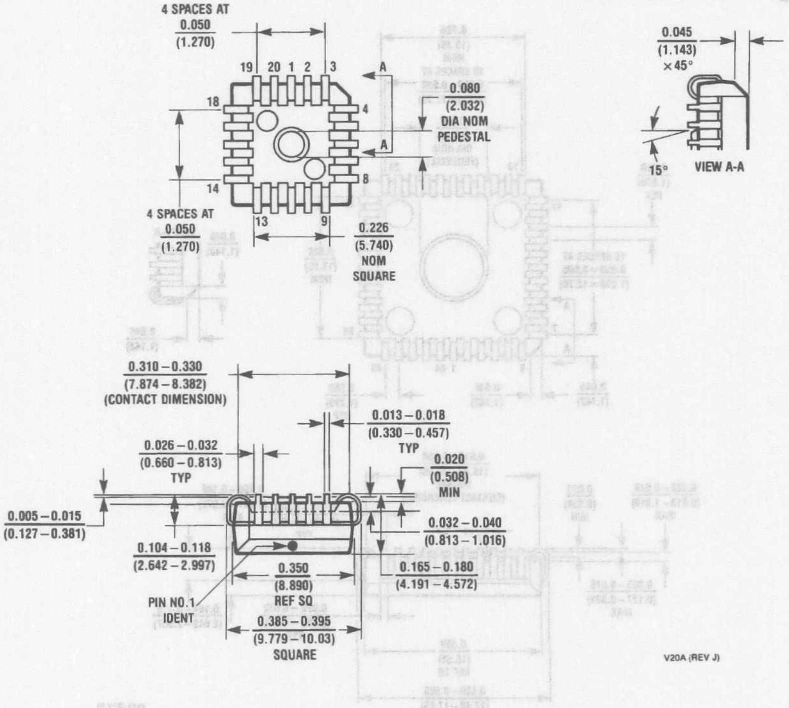


24 Lead Skinny Dual-In-Line Package (0.300″ Centers Molded) (N)
NS Package Number N24C



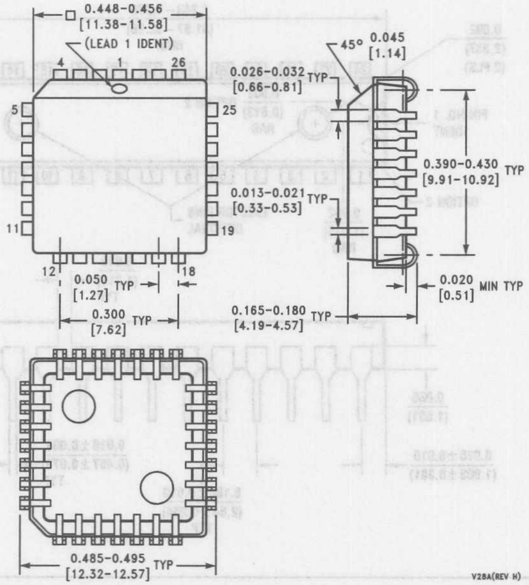
N24C (REV F)

20 Lead Plastic Chip Carrier (V)
NS Package Number V20A

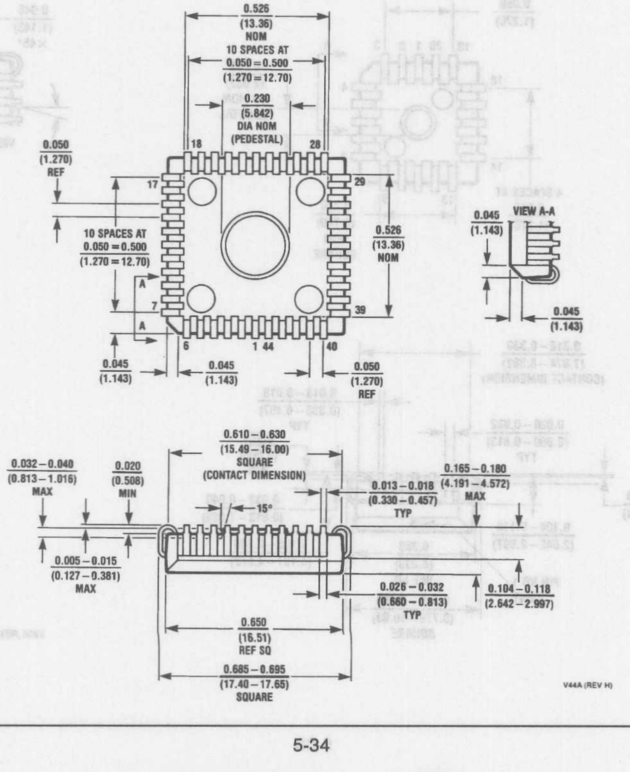


V20A (REV J)

28 Lead Plastic Chip Carrier (V)
NS Package Number V28A



44 Lead Plastic Chip Carrier (V)
NS Package Number V44A



NOTES

Bookshelf of Technical Support Information

National Semiconductor Corporation recognizes the need to keep you informed about the availability of current technical literature.

This bookshelf is a compilation of books that are currently available. The listing that follows shows the publication year and section contents for each book.

For datasheets on new products and devices still in production but not found in a databook, please contact the National Semiconductor Customer Support Center at 1-800-272-9959.

We are interested in your comments on our technical literature and your suggestions for improvement.

Please send them to:

Technical Communications Dept. M/S 16-300
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090

ALS/AS LOGIC DATABOOK—1990

Introduction to Advanced Bipolar Logic • Advanced Low Power Schottky • Advanced Schottky

ASIC DESIGN MANUAL/GATE ARRAYS & STANDARD CELLS—1987

SSI/MSI Functions • Peripheral Functions • LSI/VLSI Functions • Design Guidelines • Packaging

CMOS LOGIC DATABOOK—1988

CMOS AC Switching Test Circuits and Timing Waveforms • CMOS Application Notes • MM54HC/MM74HC
MM54HCT/MM74HCT • CD4XXX • MM54CXXX/MM74CXXX • Surface Mount

DATA ACQUISITION DATABOOK—1993

Data Acquisition Systems • Analog-to-Digital Converters • Digital-to-Analog Converters • Voltage References
Temperature Sensors • Active Filters • Analog Switches/Multiplexers • Surface Mount

DATA ACQUISITION DATABOOK SUPPLEMENT—1992

New devices released since the printing of the 1989 Data Acquisition Linear Devices Databook.

DISCRETE SEMICONDUCTOR PRODUCTS DATABOOK—1989

Selection Guide and Cross Reference Guides • Diodes • Bipolar NPN Transistors
Bipolar PNP Transistors • JFET Transistors • Surface Mount Products • Pro-Electron Series
Consumer Series • Power Components • Transistor Datasheets • Process Characteristics

DRAM MANAGEMENT HANDBOOK—1991

Dynamic Memory Control • Error Detection and Correction • Microprocessor Applications for the
DP8408A/09A/17/18/19/28/29 • Microprocessor Applications for the DP8420A/21A/22A
Microprocessor Applications for the NS32CG821

EMBEDDED CONTROLLERS DATABOOK—1992

COP400 Family • COP800 Family • COPS Applications • HPC Family • HPC Applications
MICROWIRE and MICROWIRE/PLUS Peripherals • Microcontroller Development Tools

FDDI DATABOOK—1991

FDDI Overview • DP83200 FDDI Chip Set • Development Support • Application Notes and System Briefs

F100K ECL LOGIC DATABOOK & DESIGN GUIDE—1992

Family Overview • 300 Series (Low-Power) Datasheets • 100 Series Datasheets • 11C Datasheets
Design Guide • Circuit Basics • Logic Design • Transmission Line Concepts • System Considerations
Power Distribution and Thermal Considerations • Testing Techniques • 300 Series Package Qualification
Quality Assurance and Reliability • Application Notes

FACT™ ADVANCED CMOS LOGIC DATABOOK—1990

Description and Family Characteristics • Ratings, Specifications and Waveforms
Design Considerations • 54AC/74ACXXX • 54ACT/74ACTXXX • Quiet Series: 54ACQ/74ACQXXX
Quiet Series: 54ACTQ/74ACTQXXX • 54FCT/74FCTXXX • FCTA: 54FCTXXXA/74FCTXXXA

FAST® ADVANCED SCHOTTKY TTL LOGIC DATABOOK—1990

Circuit Characteristics • Ratings, Specifications and Waveforms • Design Considerations • 54F/74FXXX

FAST® APPLICATIONS HANDBOOK—1990

Reprint of 1987 Fairchild FAST Applications Handbook
Contains application information on the FAST family: Introduction • Multiplexers • Decoders • Encoders
Operators • FIFOs • Counters • TTL Small Scale Integration • Line Driving and System Design
FAST Characteristics and Testing • Packaging Characteristics

HIGH-PERFORMANCE BUS INTERFACE DESIGNER'S GUIDE—1992

Futurebus+/BTL Devices • BTL Transceiver Application Notes • Futurebus+ Application Notes
High Performance TTL Bus Drivers • PI-Bus • Futurebus+/BTL Reference

IBM DATA COMMUNICATIONS HANDBOOK—1992

IBM Data Communications • Application Notes

INTERFACE: LINE DRIVERS AND RECEIVERS DATABOOK—1992

EIA-232 • EIA-422/423 • EIA-485 • Line Drivers • Receivers • Repeaters • Transceivers • Application Notes

LINEAR APPLICATIONS HANDBOOK—1991

The purpose of this handbook is to provide a fully indexed and cross-referenced collection of linear integrated circuit applications using both monolithic and hybrid circuits from National Semiconductor.

Individual application notes are normally written to explain the operation and use of one particular device or to detail various methods of accomplishing a given function. The organization of this handbook takes advantage of this innate coherence by keeping each application note intact, arranging them in numerical order, and providing a detailed Subject Index.

LINEAR APPLICATION SPECIFIC IC's DATABOOK—1993

Audio Circuits • Radio Circuits • Video Circuits • Display Drivers • Clock Drivers • Frequency Synthesis
Special Automotive • Special Functions • Surface Mount

LOCAL AREA NETWORK DATABOOK—1992

Integrated Ethernet Network Interface Controller Products • Ethernet Physical Layer Transceivers
Ethernet Repeater Interface Controller Products • Hardware and Software Support Products • FDDI Products • Glossary

LOW VOLTAGE DATABOOK—1992

This databook contains information on National's expanding portfolio of low and extended voltage products. Product datasheets included for: Low Voltage Logic (LVQ), Linear, EPROM, EEPROM, SRAM, Interface, ASIC, Embedded Controllers, Real Time Clocks, and Clock Generation and Support (CGS).

MASS STORAGE HANDBOOK—1989

Rigid Disk Pulse Detectors • Rigid Disk Data Separators/Synchronizers and ENDECs
Rigid Disk Data Controller • SCSI Bus Interface Circuits • Floppy Disk Controllers • Disk Drive Interface Circuits
Rigid Disk Preamplifiers and Servo Control Circuits • Rigid Disk Microcontroller Circuits • Disk Interface Design Guide

MEMORY DATABOOK—1992

CMOS EPROMs • CMOS EEPROMs • PROMs • Application Notes

OPERATIONAL AMPLIFIERS DATABOOK—1993

Operational Amplifiers • Buffers • Voltage Comparators • Instrumentation Amplifiers • Surface Mount

POWER IC's DATABOOK—1993

Linear Voltage Regulators • Low Dropout Voltage Regulators • Switching Voltage Regulators • Motion Control
Peripheral Drivers • High Current Switches • Surface Mount

PROGRAMMABLE LOGIC DEVICE DATABOOK AND DESIGN GUIDE—1993

Product Line Overview • Datasheets • Design Guide: Designing with PLDs • PLD Design Methodology
PLD Design Development Tools • Fabrication of Programmable Logic • Application Examples

REAL TIME CLOCK HANDBOOK—1991

Real Time Clocks and Timer Clock Peripherals • Application Notes

RELIABILITY HANDBOOK—1987

Reliability and the Die • Internal Construction • Finished Package • MIL-STD-883 • MIL-M-38510
The Specification Development Process • Reliability and the Hybrid Device • VLSI/VHSIC Devices
Radiation Environment • Electrostatic Discharge • Discrete Device • Standardization
Quality Assurance and Reliability Engineering • Reliability and Documentation • Commercial Grade Device
European Reliability Programs • Reliability and the Cost of Semiconductor Ownership
Reliability Testing at National Semiconductor • The Total Military/Aerospace Standardization Program
883B/RETS™ Products • MILS/RETS™ Products • 883/RETS™ Hybrids • MIL-M-38510 Class B Products
Radiation Hardened Technology • Wafer Fabrication • Semiconductor Assembly and Packaging
Semiconductor Packages • Glossary of Terms • Key Government Agencies • AN/ Numbers and Acronyms
Bibliography • MIL-M-38510 and DESC Drawing Cross Listing

TELECOMMUNICATIONS—1992

COMBO and SLIC Devices • ISDN • Digital Loop Devices • Analog Telephone Components • Software
Application Notes